Pistas Pedagógicas en Juez Online de Programación

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2015-2016
Convocatoria de Junio
Calificación: 8.5

Realizado por
Javier Martín Moreno-Manzanaro

Proyecto dirigido por
Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín

Agradecimientos

Para mí, la realización de este proyecto ha supuesto un reto así como el inicio de un camino de autoaprendizaje e investigación. Sin embargo, la satisfacción de haber alcanzado los objetivos propuestos (o al menos, algunos de ellos), compensa el esfuerzo empleado. La consecución de esa satisfacción, y la madurez adquirida en este y otros proyectos, se la debo en parte a los profesores (*casi* todos) que he tenido a lo largo de mi periplo universitario, y muy especialmente a mis directores de este Trabajo de Fin de Máster, Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín, cuya guía ha sido imprescindible.

Por último, y aunque sirva para terminar el tópico que forman estas líneas, quiero dar las gracias a mi familia. A mis padres, mi hermano y mi novia, que han ocupado los roles de mecenas, cuidadores, animadores y muchos otros con tal de brindarme apoyo en este camino hasta el final de manera incondicional.

Muchas gracias a todos.

Javier.

Resumen

Cada vez son más los usuarios que optan por usar un juez online de programación como entrenamiento para un concurso de programación o inclusive como herramienta de estudio en el ámbito docente. No obstante, es natural que con su uso se cometan errores que muchas veces no se alcanzan a entender. Por ello, en este proyecto, se trata de resolver esa carencia mediante el ofrecimiento de pistas a los usuarios. Es decir, dado un problema con un formato concreto y una solución en código fuente, se devuelve una ayuda sobre el error cometido en la solución, lo cual supone, no solo una ayuda para participantes en concursos, sino también para alumnos y profesores que pueden beneficiarse de esta ayuda. De hecho, el uso de herramientas de corrección automática está en aumento, y con la solución propuesta, se podría obtener pistas para los problemas que respeten el formato esperado.

Palabras Clave

Autoaprendizaje
Corrección automática
Desafíos de Programación
Juez Evaluador
Juez On-line
Pistas

Abstract

There are so much users who choose to use a programming online judge as a training for a real programming contest or even as a study tool in teaching. However, it is natural to make some mistakes that often they don't reach to understand. Therefore, in this project, I try to solve this gap by offering clues to users. That is, given a problem with a particular format and a solution in source code, help on the error in the solution is returned, which means not only an aid to participants in competitions, but also for students and teachers who can benefit from this aid. In fact, the use of automatic correction tools is increasing, and with the proposed solution any user could obtain clues to problems, as long as these problems respect the expected format.

Keywords

self-learning
Automatic correction
Programming Challenges
Evaluator Judge
On-line Judge
Hints

-"How many times it thundered before Franklin took the hint! How many apples fell on Newton's head before he took the hint! Nature is always hinting at us. It hints over and over again. And suddenly we take the hint"-

Robert Frost, poeta norteamericano

Índice

INTRODUCCIÓN 1

MOTIVACIÓN 1
OBJETIVOS 1
REPERCUSIÓN 2
PLAN DE TRABAJO 2
ESTRUCTURA DEL DOCUMENTO 3

INTRODUCTION 4

MOTIVATION 4
GOALS 4
REPERCUSSION 5
WORKPLAN 5
DOCUMENT STRUCTURE 5

ESTADO DEL ARTE 6

JUECES DE PROGRAMACIÓN ON-LINE 6
UVA ONLINE JUDGE 9
LEETCODE 10
CODECHEF 12
HACKEREARTH 14
SPHERE ONLINE JUDGE 17
CODEFORCES 19
¡ACEPTA EL RETO! 21

PUNTO DE PARTIDA 24

DEFINICIÓN DE LOS EJERCICIOS 24
HINTEXTRACTOR 30

IMPLEMENTACIÓN DE PISTAS 31

TOMA DE DECISIONES **31**
TIPOS DE PISTAS **33**
PISTAS FINALES **34**
RUNTIMEERROR 35
TIMELIMITEXCEEDED 37
WRONGANSWER 37

EVALUACIÓN Y ESTADÍSTICAS **39**

ANÁLISIS DE LOS RESULTADOS **39**
EJEMPLOS **42**
FALLO EN EL EMPTY 42
FALLO EN EL SAMPLE 44
RTE SIN DEPENDENCIA 46
WA SIN DEPENDENCIA 48

CONCLUSIONES **50**

VALORACIÓN PERSONAL **50**
TRABAJO FUTURO **51**
INTEGRACIÓN EN EL JUEZ DE ¡ACEPTA EL RETO! 51
EVALUACIÓN EXHAUSTIVA PARA TLE 52
INTEGRACIÓN CON EJERCICIOS INTERACTIVOS 52
PISTAS BASADAS EN LA COMUNIDAD 52
PISTAS EN BASE A LA DIFICULTAD DEL PROBLEMA 53

CONCLUSIONS **54**

PERSONAL RATING **54**
FUTURE WORK **55**
INTEGRATION IN ¡ACEPTA EL RETO! 55
COMPREHENSIVE ASSESSMENT FOR TLE 56
INTEGRATION WITH INTERACTIVE EXERCISES 56
COMMUNITY-BASED HINTS 56
HINTS BASED ON THE DIFFICULTY OF THE PROBLEM 57

BIBLIOGRAFÍA **58**

Introducción

Motivación

El uso de jueces online de programación aumenta a medida que pasa el tiempo, y es cada vez más común su uso no solo como entrenamiento para concursos, sino como una parte del conjunto de herramientas que un alumno tiene a su disposición para prepararse académicamente y mejorar sus habilidades de programación. Estos jueces evalúan un problema que prueban con distintos datos de entrada, generalmente recogidos en un fichero .in y devuelven el resultado del envío tras comparar las salidas generadas con un conjunto de salidas de referencia que posee el juez, generalmente recogidas en un fichero .out. Sin embargo el resultado devuelto es prácticamente binario: la solución está bien o está mal. Así pues, con esto en mente, resulta necesario procurar que los usuarios que envían soluciones a los problemas planteados tengan algo de información cuando cometen un error que les permite comprender qué ha sucedido y por tanto, les ayude a identificar sus propios errores. De hecho, el juez sobre el que trabajaremos (¡Acepta el reto!), es utilizado en varias asignaturas como elemento pedagógico.

Por todo esto, decidí embarcarme en la realización de este proyecto y así desarrollar una herramienta que será aprovechada por un gran número de personas.

Objetivos

Se pretende conseguir un mecanismo con el que se ofrezca a los usuarios alguna pista acerca de los errores que cometen en sus envíos. Sin embargo, estas pistas constituirán un punto de partida y no una solución en sí mismas, de manera que los usuarios sepan por dónde empezar a indagar, para depurar su código y poder alcanzar así una solución válida. Así, al margen del hecho de conseguir llegar a un resultado válido, lo importante es que logren llegar a ese punto por sí mismos, aprovechando el pequeño empujón que supone la pista ofrecida.

Además, las pistas serán de distintos tipos en función del veredicto arrojado por el juez y se pretende ayudar a depurar una solución, realizando ciertas pruebas como por ejemplo la comprobación de la existencia de una dependencia entre los casos probados que sea ocasionada por el código enviado; si la solución es correcta pero se escribe de más; si en caso de alcanzar el límite de tiempo máximo permitido para el problema, la salida generada era correcta y por tanto es una cuestión de optimización de código;

o si por el contrario el usuario ya iba mal, y el hecho de superar el umbral de tiempo establecido es un error más.

Con esto se conseguirá dar información, que no es evidente a simple vista, y que sin dar la solución hecha, ayuda al usuario a comprender cuáles son los problemas del código que ha enviado

Repercusión

Cabe destacar además, que atendiendo a la repercusión académica, este proyecto se plantea como una ayuda tanto para el aprendizaje de los alumnos como para la labor docente de los profesores, pues manteniendo un formato de ejercicio concreto, está funcionalidad se mantiene independiente del juez evaluador utilizado, ya que para extraer la pista, se realizarán las ejecuciones y comparaciones pertinentes de forma independiente.

Ahora bien, si pensamos por ejemplo, en su integración con el juez de Acepta el Reto, que es usado por varios profesores en sus asignaturas de programación como herramienta de evaluación, podemos tener una cierta idea de la utilidad del proyecto. Algunas de las asignaturas en las que se utiliza son:

- Fundamentos de la programación
- Estructuras de Datos y Algoritmos
- Diseño de Algoritmos
- Métodos Algorítmicos en la Resolución de Problemas

Por tanto, se puede considerar esta herramienta, no sólo como una ayuda en la preparación de concursos de programación, sino también como herramienta pedagógica en el aprendizaje o perfeccionamiento de conceptos de programación y su revisión por parte de un profesor.

Plan de trabajo

Al inicio del proyecto se plantea un calendario de trabajo con una semana como medida de tiempo para revisar los avances logrados mediante la celebración de una reunión con los directores del proyecto. En cada una de estas reuniones se expone lo que se ha hecho durante esa semana y los problemas que se han encontrado. Asimismo, se proponen una serie de objetivos a conseguir durante la semana siguiente para, de nuevo, exponerlos en una reunión.

Es decir, la planificación del presente proyecto se sustenta sobre los fundamentos básicos de la metodología ágil SCRUM [9], no obstante, su aplicación es más flexible pues sólo se cumplen algunas de las propuestas de la citada metodología, por razones logísticas.

Estructura del documento

Así pues, tras esta introducción se dedica un capítulo a exponer cómo tratan otros jueces las pistas para envíos y qué forma tienen, para después plantear cuál era el estado del proyecto al inicio y de qué se disponía. Posteriormente se expondrá el trabajo realizado, y se mostrarán también algunos resultados y estadísticas obtenidas a partir de varias evaluaciones, para finalizar, por último, con un apartado de conclusiones con una valoración personal y propuestas de trabajo futuro.

Introduction

Motivation

The use of programming online judges increases as time passes, and it is increasingly common use not only as training for competitions, but as part of the toolkit that a student has available to prepare academically and improve his programming skills. These judges evaluate a problem that is tested with different input data, usually collected in a file .in and return the execution result after comparing the generated outputs with a set of reference outputs that the judge has, usually contained in a file .out. However the result returned is virtually binary: the solution is right or wrong. So, with that in mind, it is necessary to ensure that users who send solutions to problems have some information when they make a mistake that allows them to understand what has happened and therefore help them identify their own mistakes. In fact, the judge on which I will work (¡Acepta el reto!), is used in various subjects like educational element.

For all this, I decided to embark on this project and develop a tool that will be used by a large number of people.

Goals

It aims to achieve a mechanism by which users are offered a clue about the mistakes in their shipments. However, these hints constitute a starting point and not a solution by themselves, so that users know where to begin to investigate, to debug their code and can thus achieve a valid solution. Thus, apart from the fact of getting a valid result, it is important to make users get to that point by themselves, taking advantage of the little push that represents the offered hint.

In addition, the clues are of different types depending on the verdict thrown by the judge and is intended to help to debug a solution, performing certain tests such as checking the existence of a dependency between proven cases that is caused by the sent code; If the solution is correct but has written more than expected; if in case of reaching the maximum time limit allowed for the problem, the generated output was correct and therefore is a matter of code optimization; or if instead the user already was wrong, and the fact of exceed the threshold of time is a mistake more.

With this tool, I will achieve to give some information, which is not obvious to the naked eye, and without giving the final solution, in order to help the user to understand what the problems of the sent code are.

Repercussion

It should also be noted, that in response to the academic impact, this project is presented as an aid to both student learning and for educational labor of teachers, as maintaining a particular format for the exercises, this functionality remains independent from the evaluator judge used, because to extracting the hints, relevant executions and comparisons will be made in an independent way.

Now, if we think for example in its integration with the judge ¡Acepta el reto!, which is used by several teachers in their subjects programming as an evaluation tool, we can have some idea of the usefulness of the project. Some of the subjects in which the judge is used are:

- Programming Basics
- Data Structures and Algorithms
- Programming Technology
- Algorithmic Methods in Problem Solving

Therefore, this tool can be considered not only as an aid in the preparation of programming contests, but also as an educational tool in learning or improving programming concepts and in the review by a teacher.

Workplan

At the beginning of the project, a work schedule was posed with a week as a measure of time to review the progress achieved by celebrating a meeting with project managers. In each of these meetings we discussed what has been done during that week and the problems found. Also, a number of objectives are proposed to achieve during the following week to again expose the experience in the next meeting.

That is, the planning of this project is based on the fundamentals of agile methodology SCRUM [9], however, its application is more flexible because only some of the proposals in this methodology are met, for logistical reasons.

Document Structure

So, after this introduction, a chapter is dedicated to exposing how other judges treat hints for the user's shipments and how they are, then I will expose what was the status of the project at the beginning and what was available for me. Then the work made will be presented and some results and statistics obtained are also shown from several evaluations, for, finally, end with a section of conclusions with a personal assessment and proposals for future work.

Estado del arte

Para poder realizar un análisis sobre la forma y el contenido de las pistas que se dan a los usuarios sobre sus envíos de problemas, se ha realizado un estudio inicial repasando este aspecto en distintos jueces evaluadores on-line. Resulta destacable la circunstancia de que esta característica no se encuentra disponible en todos los jueces pues depende un poco de la "política" más o menos estricta que tenga cada uno.

Sin embargo, antes de comenzar a profundizar en cada uno de los jueces, se debe tener claro el concepto de juez on-line y su funcionamiento.

Jueces de programación on-Line

Un juez on-line de programación es una herramienta web que pone a disposición de sus usuarios un conjunto de problemas de programación, generalmente de distintas temáticas y dificultades. Los interesados, pueden elegir uno de estos problemas, y tratar de resolverlo enviando su código al juez. Estos problemas, suelen tener algunas restricciones, como el hecho de que no se plantean problemas con soluciones en un entorno gráfico, sino solamente con interacción por consola haciendo uso de la entrada y la salida estándar. En los enunciados se suele explicar en qué consiste el problema, el formato de la entrada que se espera recibir, y el formato de salida que se debe generar, además de algunos ejemplos para ilustrar dichos formatos, tal y como se puede apreciar en el enunciado presentado en la página siguiente. Una vez el usuario considere su código acabado, puede enviarlo usando la plataforma de envío[1] del juez y este comenzará su evaluación.

Lo primero es compilar la solución enviada. Si la solución no compila, el juez no podrá ejecutarla, por lo que parará su evaluación, en caso contrario, comenzará a hacer pruebas con ella. Para realizar estas pruebas, los jueces mantienen para cada problema un conjunto de casos de prueba con distintos datos de entrada para testear la calidad de la solución. Dichos casos suelen recogerse en un fichero .in del que se irán leyendo para generar las salidas oportunas. Estas salidas serán recogidas en un fichero .out en el que se irán acumulando.

[1] La plataforma de envío suele variar según el diseño web y las opciones que se permitan (añadir comentarios, resaltado de sintaxis, etc.), no obstante, en la mayoría se puede realizar el envío escribiendo directamente en un campo habilitado para ello o enviando ficheros con el código fuente.

Problema número 100
http://www.aceptaelreto.com/problem/statement.php?id=100

Constante de Kaprekar

Tiempo máximo: 2,000 s Memoria máxima: 4096 KiB

El matemático indio Dattaraya Ramchandra Kaprekar descubrió en 1949 una curiosa característica del número 6174. Hoy, se conoce a dicho número como *constante de Kaprekar* en honor a él.

El número es notable por la siguiente propiedad:

1. Elige un número de cuatro dígitos que tenga al menos dos diferentes (es válido colocar el dígito 0 al principio, por lo que el número 0009 es válido).
2. Coloca sus dígitos en orden ascendente y en orden descendente para formar dos nuevos números. Puedes añadir los dígitos 0 que necesites al principio.
3. Resta el menor al mayor.
4. Vuelve al paso 2.

A este proceso se le conoce como *la rutina de Kaprekar*, y *siempre* llegará al número 6174 en, como mucho, 7 iteraciones. Una vez en él, el proceso no avanzará, dado que 7641 − 1467 = 6174.

Por ejemplo, el número 3524 alcanzará la constante de Kaprekar en 3 iteraciones:

5432 − 2345 = 3087
8730 − 0378 = 8352
8532 − 2358 = **6174**

Los únicos dígitos de cuatro cifras para los que la rutina de Kaprekar *no* alcanza el número 6174 son los *repdigits*, es decir aquellos cuyas cuatro cifras son iguales (como 1111), pues en la primera iteración se alcanzará el valor 0 y no podrá salirse de él. Es por esto que en el paso 1 se pedía explícitamente que el número inicial tuviera al menos dos dígitos diferentes.

El resto de los números de cuatro cifras terminarán siempre en el número 6174.

A continuación se muestran dos ejemplos más:

- El número 1121 necesita 5 iteraciones:

2111 − 1112 = 0999
9990 − 0999 = 8991
9981 − 1899 = 8082
8820 − 0288 = 8532
8532 − 2358 = **6174**

- El número 1893 necesita 7:

9831 − 1389 = 8442
8442 − 2448 = 5994
9954 − 4599 = 5355
5553 − 3555 = 1998
9981 − 1899 = 8082
8820 − 0288 = 8532
8532 − 2358 = **6174**

Entrada

La primera linea de la entrada contendrá el número de casos de prueba. Cada uno contendrá, en una única línea, un número a comprobar.

Salida

Para cada caso de prueba, el programa indicará el número de vueltas que se debe dar a la rutina de Kaprekar para alcanzar el 6174. Para los números *repdigits* deberá escribir 8. Para la propia constante de Kaprekar deberá indicar 0.

Entrada de ejemplo

```
5
3524
1111
1121
6174
1893
```

Salida de ejemplo

```
3
8
5
0
7
```

Autores: Pedro Pablo Gómez Martín; Patricia Díaz García; Marco Antonio Gómez Martín

Enunciado 1

Además, ya que estos jueces toman su concepto original como preparación para concursos, cada problema suele definir ciertas condiciones adicionales en cuanto al uso de memoria que se emplea en la solución, y al tiempo invertido en resolver el problema. Por ello, si durante la ejecución de la solución enviada, alguno de estos límites es superado, aunque la salida generada fuera correcta, se considerará la solución como no válida.

De igual forma, para cada problema debe existir una salida de referencia. Dependiendo del juez, el problema podrá contener solamente dicha salida, o el código fuente de la solución que la genera. Independientemente de los elementos que se decidan incluir o no en el problema, lo normal es que dado que esa solución es la que se usa como referencia y va a ser siempre la misma, se ejecutará una única vez y se trabajará en adelante simplemente con la salida.

Así pues, teniendo las salidas de referencia y las de la solución enviada, sólo resta compararlas y comprobar si el resultado coincide o no. Al resultado de esa evaluación es común denominarla veredicto, y cada juez dispone de los suyos propios. En general, aunque estos veredictos tengan ciertos matices diferentes de un juez a otro, e incluso algunos definan más o menos veredictos para expresar situaciones concretas, es cierto que hay un conjunto de ellos, que pretenden expresar las mismas ideas. Los más comunes son:

- Error en la compilación
- Error en tiempo de ejecución
- Límite de tiempo superado
- Salida generada distinta de la esperada
- Problema aceptado. La comparación de las salidas es igual y no se supera el límite de tiempo.

Finalmente, el usuario conocerá el veredicto del juez y podrá volver a realizar un nuevo envío en caso de no haber conseguido un veredicto favorable, aunque de nuevo, esto vuelve a ser un criterio voluble, puesto que cada juez puede definir sus propios criterios al respecto: número de intentos por problema, disponibilidad de los problemas, etc.

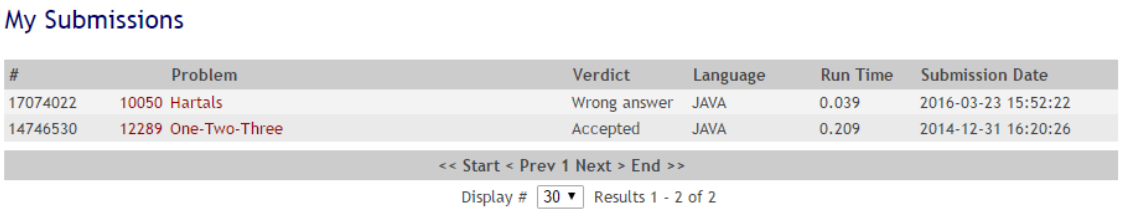
Así pues, ahora que se ha introducido el concepto básico de un juez online de programación, y su funcionamiento general, vamos a hacer un repaso de aquellos de especial relevancia para el estudio que nos ocupa, fijándonos sobre todo, en el aspecto del ofrecimiento de pistas a envíos erróneos de los usuarios.

UVa Online Judge

Impulsado por la universidad de Valladolid, el juez de la UVa [2] nació de la mano de Miguel Ángel Revilla, profesor de algoritmia en dicha universidad, en el año 1995, aunque no estaría disponible para el público hasta un par de años más tarde. Pasados algunos años, ya en 2007, Miguel Ángel acometería una profunda remodelación del sistema, sustituyendo el anterior, cuya primera versión fue desarrollada en primera instancia por un alumno, Ciriaco García de Celis; por un nuevo sistema desarrollado por él mismo y un nuevo servidor.

Este juez mantiene una base de usuarios registrados de más de 800.000 y una cantidad de problemas que supera los 4.000, motivos que junto a su veteranía constituyen sus puntos fuertes y son lo que lo convierten en uno de los jueces de más relevancia.

Sin embargo, pese a su notoriedad en el mundillo, en lo referente a las pistas este juez tiene una clara carencia ya que no otorga ningún tipo de indicación para los usuarios, al menos, no por sí mismos, ya que recientemente han incorporado aplicaciones de terceros, que te ayudan a obtener un conjunto de casos de prueba, que te sirva para obtener una salida correcta. No obstante este tipo de ayudas, siguen sin ser pistas sobre el propio envío, y como resultado simplemente se obtiene un veredicto u otro.


My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
17074022	10050 Hartals	Wrong answer	JAVA	0.039	2016-03-23 15:52:22
14746530	12289 One-Two-Three	Accepted	JAVA	0.209	2014-12-31 16:20:26

<< Start < Prev 1 Next > End >>

Display # 30 Results 1 - 2 of 2

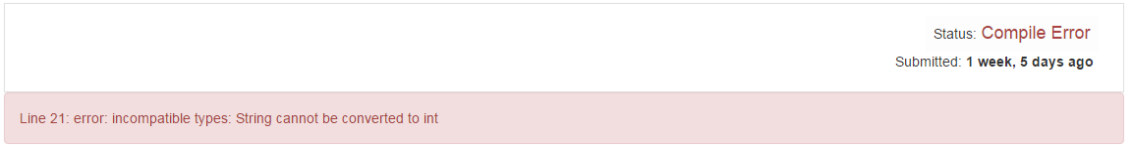
Uva 1

A pesar de esto, ya que lo único que devuelve es el veredicto final, sí que mantiene una página en la que se da una explicación, un tanto escueta, eso sí, de todos y cada uno de los posibles veredictos o estados que puede tener un envío. Por lo tanto, lo más cercano a una pista que ofrece este juez, es la salida del compilador en caso de obtener un error durante la compilación.

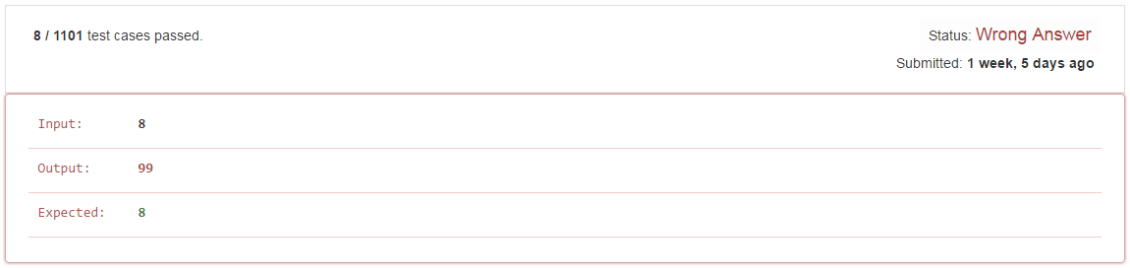
LeetCode

Ubicada en California, LeetCode [8] es una empresa privada que ofrece problemas de programación que ciertas empresas han puesto como prueba en alguna ocasión durante un proceso de selección para un puesto de trabajo [1]. Dispone, de hecho, de problemas etiquetados por empresas (Google, Amazon, Dropbox o Snapchat, entre muchas), aunque esta funcionalidad sólo está disponible para usuarios Premium (de pago) así como ciertos problemas y ciertos artículos en los que se explica detalladamente la solución a algunos ejercicios.

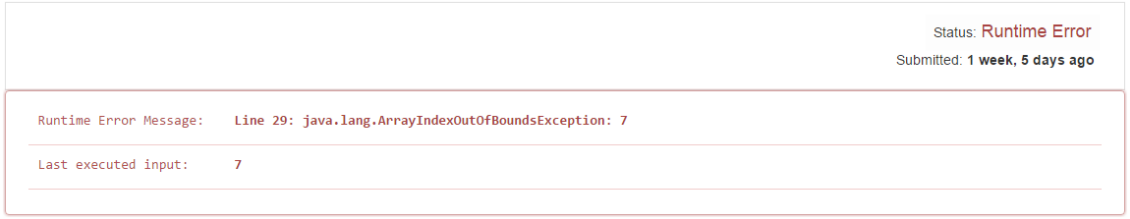
Tras algunas pruebas con la versión gratuita de este juez, se puede apreciar la forma con la que se presentan las pistas. En todos los veredictos se ofrece algo de información adicional, que varía en función de la resolución que se haya devuelto. Por ejemplo, para un W*rong Answer* se devuelve la entrada del caso de prueba, la salida generada y la salida esperada; y para un *Runtime Error* se devuelve la entrada del último caso de prueba ejecutado y el error (excepción) generado.



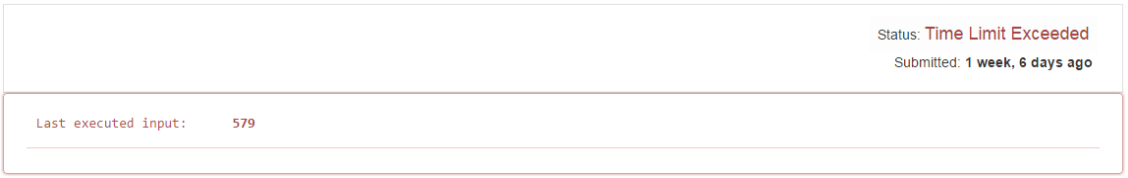
LeetCode 1



LeetCode 2



LeetCode 3



LeetCode 4

En este juez, la pista que se devuelve se refiere al primer caso de prueba que devuelve un error. Además, quizá por su carácter empresarial y comercial, no ofrece demasiada información acerca de los posibles veredictos existentes, ni del significado o alcance de cada uno, así como tampoco se menciona nada acerca de la prioridad de los mismos al hacer las evaluaciones. No obstante, tras las pruebas realizadas, se ha identificado la siguiente relación:

Compile Error > Runtime Error > Time Limit Exceeded > Wrong Answer > Accepted

Donde el símbolo '>' significa en este caso *'se comprueba antes que'*.

CodeChef

CodeChef [3] es un juez online de origen indio, promovido por otra empresa de Internet, Directi, pero esta vez con propósitos educativos, tratando de conseguir acercar el mundo de la programación a los jóvenes. Uno de sus principales objetivos era el de conseguir que estudiantes de secundaria consiguieran un cierto manejo en cuanto a habilidades de programación se refiere, para así poder destacar en la llamada Olimpiada Internacional de Informática, aunque en la actualidad se orienta tanto a estudiantes como a profesionales disponiendo de problemas de bastantes niveles de dificultad.

Aunque en esta ocasión, el portal web del juez ofrece más información (sobre todo en el foro), sigue existiendo cierta carencia en cuanto a recoger y explicar los distintos tipos de veredictos que se manejan. Además, en caso de no conseguir un aceptado como resultado de un problema, la información que un usuario recibe es cuanto menos confusa, pues existe una división entre tareas y subtareas que no se explica, al menos en primera instancia, ya que existe un programa (remunerado) para gente que quiera publicar problemas para el que es necesario enviar un solicitud, y aportar una serie de datos.

Asimismo, tampoco se dispone de información acerca de las entradas o salidas generadas y/o esperadas, sino tan sólo una tabla resumen indicando un veredicto para cada tarea (conjunto de casos de prueba), sin ningún enlace, ni elemento con el que poder interactuar, para saber más.

Submission Info:

Sub-Task	Task #	Score	Result (time)
1	0	NA	AC (0.080000)
1	1	NA	TLE (2.010000)
Final Score - 0.000000			Result - TLE
2	2	NA	TLE (2.010000)
2	3	NA	TLE (2.010000)
Final Score - 0.000000			Result - TLE
3	4	NA	TLE (2.010000)
3	5	NA	TLE (2.010000)
3	6	NA	TLE (2.010000)
Final Score - 0.000000			Result - TLE

CodeChef 1

Cabe destacar también que este juez dispone de algunos veredictos para errores muy concretos, que pueden suponer una ayuda más que notable,

pues a pesar de no ser una pista como tal, orienta en gran medida al usuario. Dichos veredictos son los siguientes:

VEREDICTO	SIGNIFICADO	CAUSAS
SIGSEGV	Referencia de Memoria no válida	• Uso excesivo de memoria. • Acceso fuera de rango a un array.
SIGABRT	Error Fatal	• Sentencia assert en C++ que devuelve false. • Elementos de la STL almacenando demasiada memoria.
SIGFPE	Error de Punto Flotante	• División (o módulo) por 0
NZEC	No se devuelve código de terminación 0	• En C, el método main no tiene un return 0. • En C++/Java puede ser por una excepción.

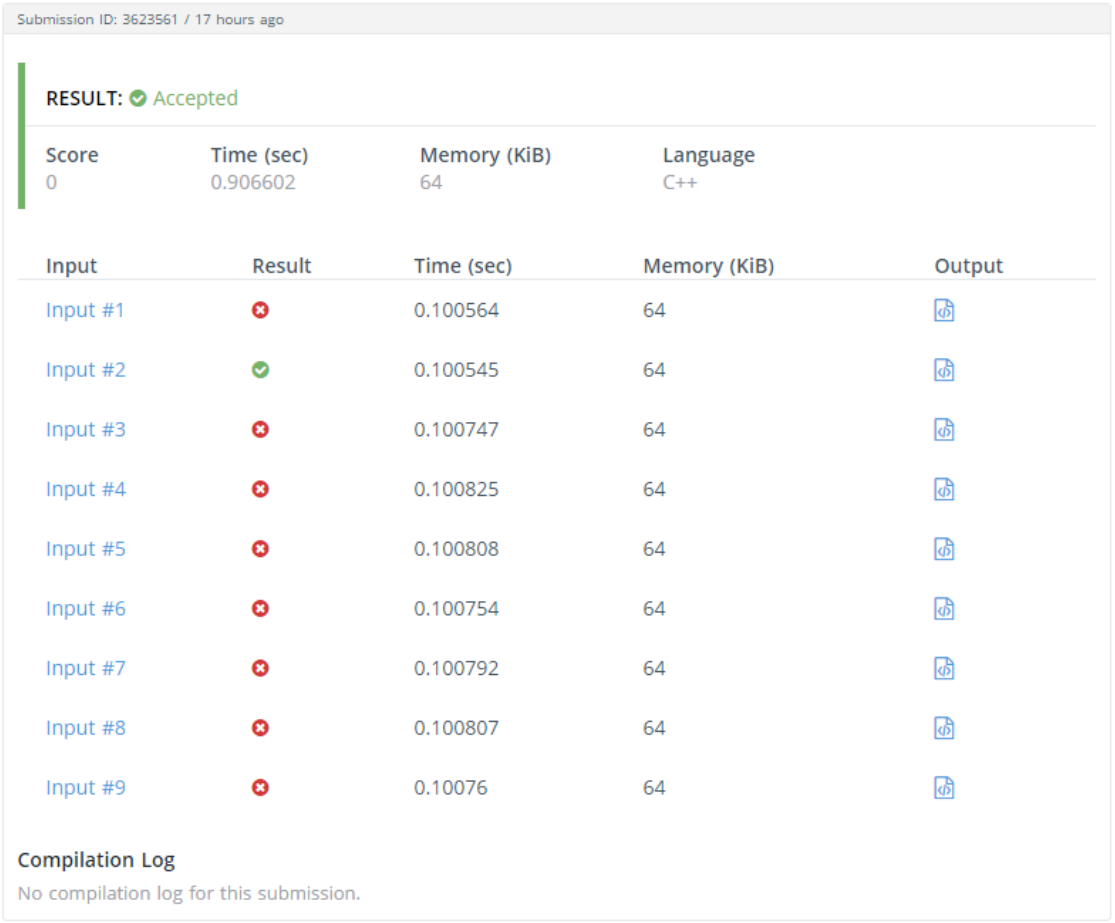
Estos veredictos son explícitos para los casos citados, pero son englobados bajo el veredicto de *Runtime Error.*

Esta circunstancia, es quizá heredada por el uso del juez SPOJ (Sphere Online Judge), del que hablaremos más adelante, puesto que en la web de codeChef, se explica que utiliza SPOJ, como backend. Sin embargo no queda muy claro, de qué forma, ya que en su documentación acerca de los veredictos, existen algunas sutiles diferencias tal y como veremos en apartados siguientes.

HackerEarth

Nos encontramos, de nuevo, ante un juez de origen indio. HackerEarth [4] es una empresa dedicada a proveer soluciones para la contratación de personal, de manera que aunque el uso del juez es gratuito, tiene un propósito lucrativo.

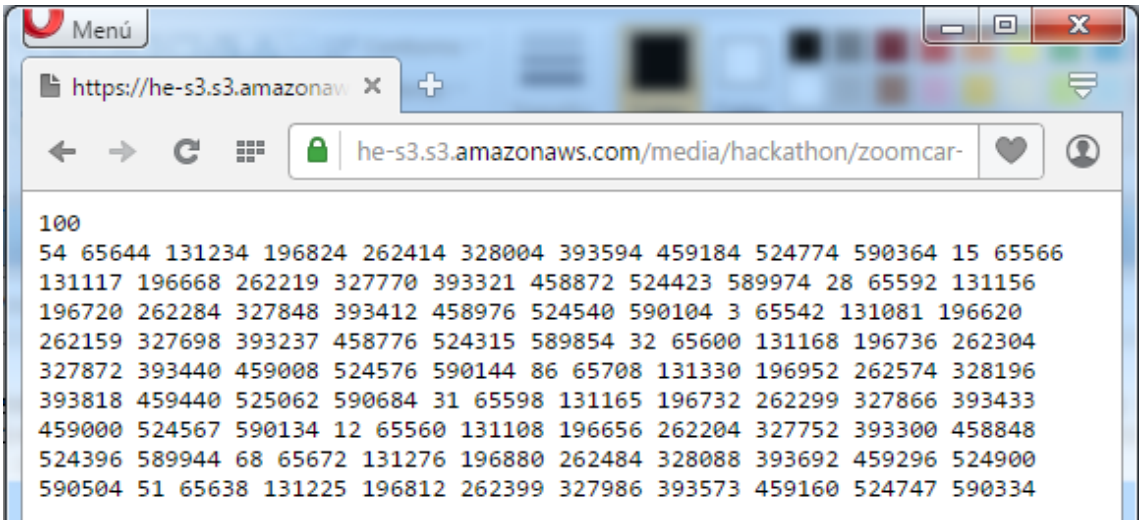
Este juez posee una cualidad un tanto peculiar pues permite tener envíos parcialmente correctos, es decir, envíos en los que se pasa alguno de los tests, pero no todos. Esta medida, es llevada a cabo como empujón motivador para los usuarios que empiezan, pero no deja de ser una decisión, que te permite marcar como aceptado un envío que en realidad no lo es.



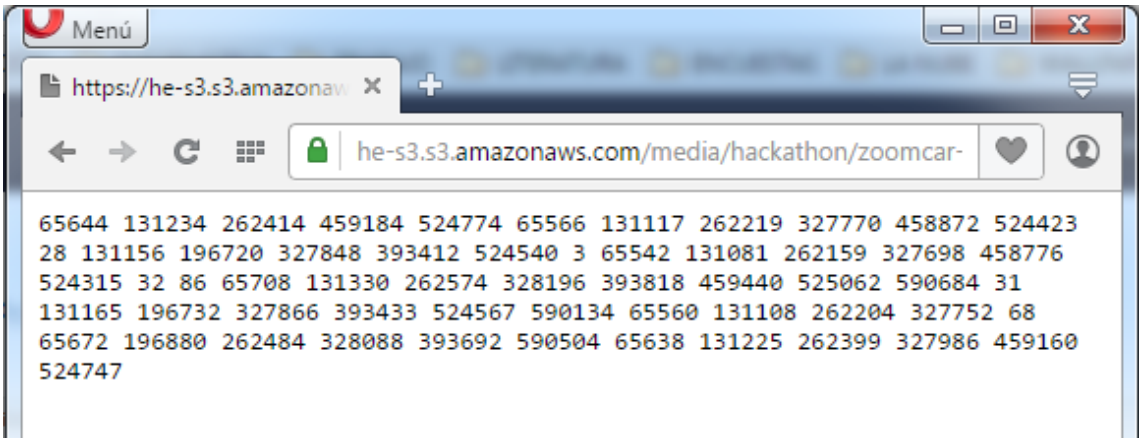
HackerEarth 1

Como se aprecia en la imagen anterior, de todos los conjuntos de entradas sólo uno, es marcado como aceptado, siendo el resto, en este caso *wrong answer,* sin embargo, el resultado que se ofrece es *accepted.* Aunque usan también un sistema de puntuación, y en este caso, se otorgan cero puntos, puede que esta política induzca a malos entendidos.

No obstante, para compensar esta característica, las pistas que se le ofrecen al usuario, son de mucha más ayuda ya que directamente se le ofrece la posibilidad, de descargar cada fichero de entrada y de salida usados durante la evaluación del problema enviado.



HackerEarth 2



HackerEarth 3

De esta manera, no se indica la entrada concreta con la que la ejecución del código enviado falló, pero se da la posibilidad de que el usuario reproduzca localmente, la evaluación exacta que el juez hace, y pueda incluso depurarlo paso a paso.

Por otro lado, este juez también da algunos veredictos más específicos según el resultado de la ejecución, aportando así un extra más de información. Estos veredictos se recogen en la siguiente tabla:

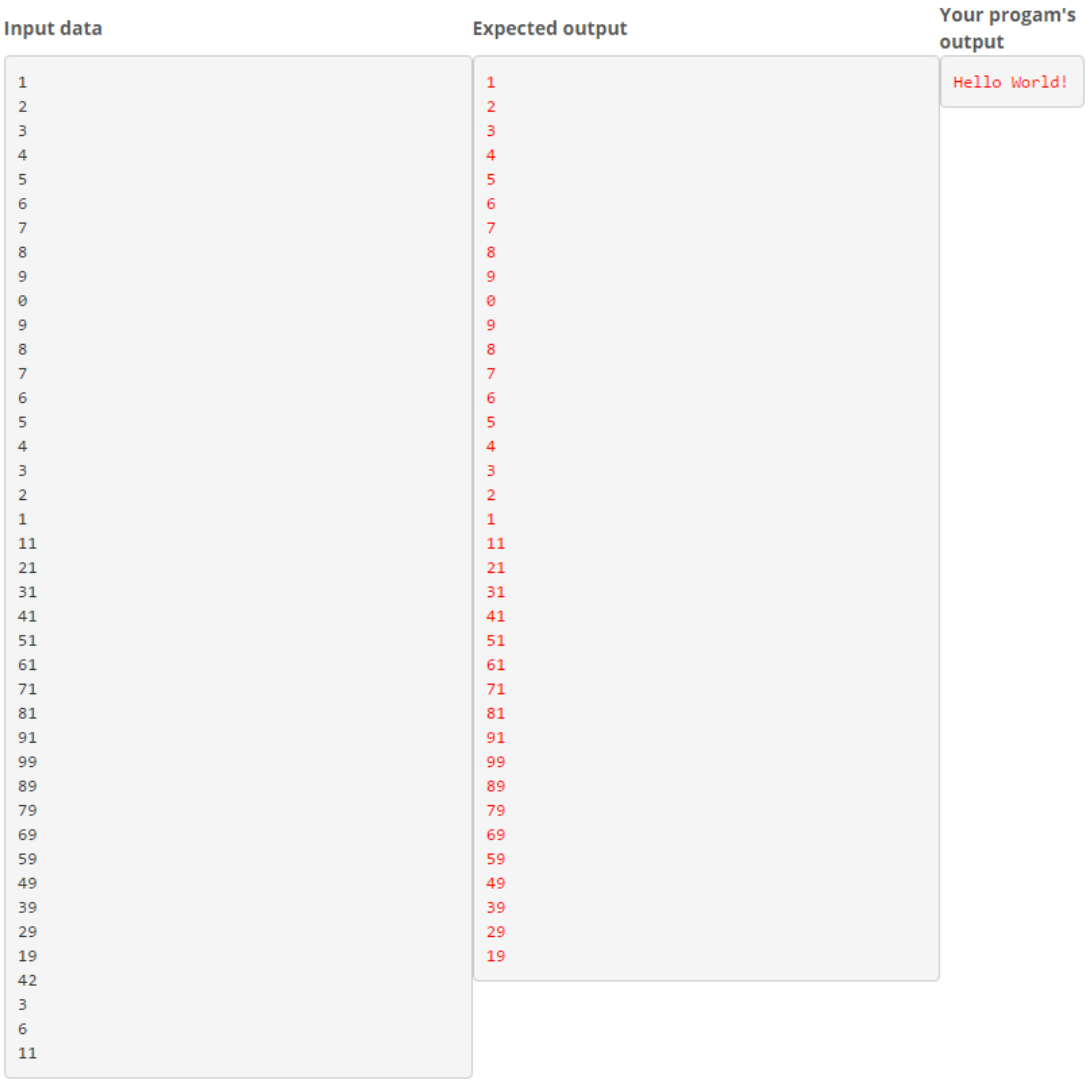
VEREDICTO	SIGNIFICADO	CAUSAS
SIGSEGV	Fallo de segmentación	• Uso excesivo de memoria. • Acceso fuera de rango a un array. • Puntero inicializado incorrectamente
SIGXFSZ	Límite de salida superado	• El envío escribe demasiados datos en la salida
SIGABRT	Error Fatal	• Memoria insuficiente
SIGFPE	Error de Punto Flotante	• División (o módulo) por 0 • Raíz cuadrada de número negativo
NZEC	No se devuelve código de terminación 0	• El método main no tiene un return 0. • El código lanza una excepción no controlada. • Uso de una librería externa que tiene algún error
OTHER	Otros problemas de uso de memoria	• Uso de demasiada memoria • Arrays u otros elementos que crecen demasiado en tamaño • Causas de SIGSEGV

Sphere Online Judge

Originario de Polonia, SPOJ [5] tiene detrás a la empresa Sphere Research Labs. De manera análoga a algunos de los jueces que ya hemos visto, el uso del juez es gratuito y está abierto al público, aunque su actividad comercial viene de la integración de sus servicios en otras aplicaciones.

En cuanto a las pistas ofrecidas, y tras una serie de pruebas realizando algunos envíos, sólo recibimos indicaciones para los resultados de *wrong answer*. En caso de *compilation error*, se muestra la salida del compilador, pero para errores de *time limit, runtime error*, etc; lo máximo que se recibe es un enlace para volver a probar el código enviado en un IDE aislado del que dispone SPOJ.

Además, para los casos de *wrong answer,* no siempre se ofrece la posibilidad de tener pistas sobre nuestro envío, sino que esto depende del problema realizado (probablemente, de la dificultad del mismo, dando pistas sólo en



Sphere Online Judge 1

los más básicos). Así pues, la información ofrecida en estos casos, es una muestra de la entrada, la salida esperada y la salida generada por el envío.

A pesar, de no dar pistas en todos los problemas, SPOJ tiene, al igual que los anteriores jueces, veredictos concretos para los casos de *runtime error*. Dichos veredictos se recogen en la siguiente tabla:

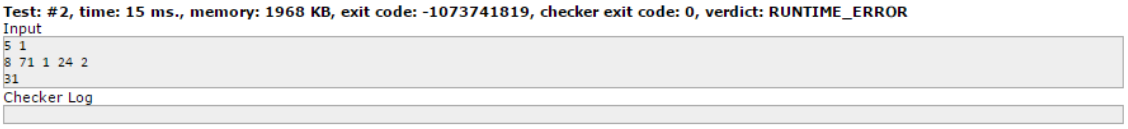
VEREDICTO	SIGNIFICADO	CAUSAS
SIGSEGV	Fallo de segmentación	• Acceso fuera de rango a un array. • Puntero inicializado incorrectamente
SIGXFSZ	Límite de salida superado	• El envío escribe demasiados datos en la salida (más de 25MB)
SIGABRT	Error Fatal	• Memoria insuficiente • STL de C++ almacenando demasiada memoria. • Llamada del sistema abort()
SIGFPE	Error de Punto Flotante	• División (o módulo) por 0 • Raíz cuadrada de número negativo
NZEC	No se devuelve código de terminación 0	• El método main no tiene un return 0. • En lenguajes interpretados (y JAVA), el código lanza una excepción no controlada.
OTHER	Resto de problemas	-----

CodeForces

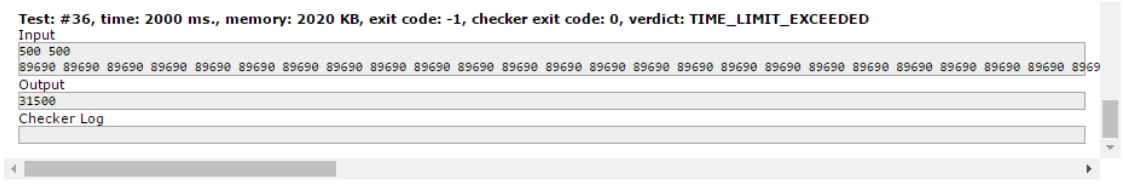
CodeForces [6], de origen ruso, está dedicado a la realización de concursos de programación, con distintos tipos de reglas, habiendo acogido por ejemplo el Yandex Algorithm Contest en un par de ocasiones, que es un concurso promovido por la empresa Yandex, conocida habitualmente como "el Google Ruso". En este caso, la creación y mantenimiento del juez no depende de una empresa, sino de miembros de la universidad estatal de Saratov que se encargan de ello con la única intención de ofrecer una herramienta para practicar la programación.

La documentación de este juez es algo escasa en cuanto a tipos de veredictos y orden de prioridad entre ellos, aunque tras algunas pruebas de envíos, se puede identificar un aspecto que si bien no está documentado en ninguna página (al menos fácilmente accesible) difiere de lo visto en otros jueces. Esta característica es la de parar la evaluación al encontrar un primer fallo (independientemente del veredicto), y no realizar los tests posteriores. Además aunque no dispone de una documentación al uso que explique los distintos veredictos que maneja, se pueden ver los envíos que están sucediendo en tiempo real, y de esa tabla se puede sacar la conclusión de que utilizan los usados como estándar de facto (*Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, Runtime Error* y *Compilation Error*)

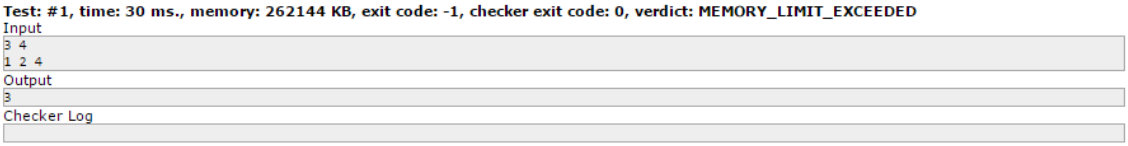
Centrándonos ya en las pistas ofrecidas sobre los envíos, se utiliza el mismo esqueleto independientemente del veredicto final del juez. Se muestra para cada test ejecutado, la entrada leída, la salida generada por el código enviado, la salida esperada, y un log acerca del error encontrado para los casos de wrong answer; mostrando estos elementos según proceda y acorde al veredicto.



Codeforces 1

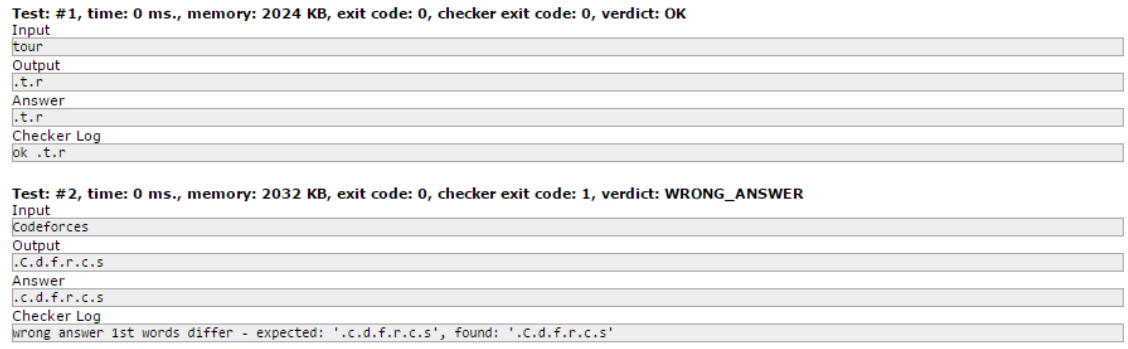


Codeforces 2



Codeforces 3

→Judgement Protocol



Codeforces 4

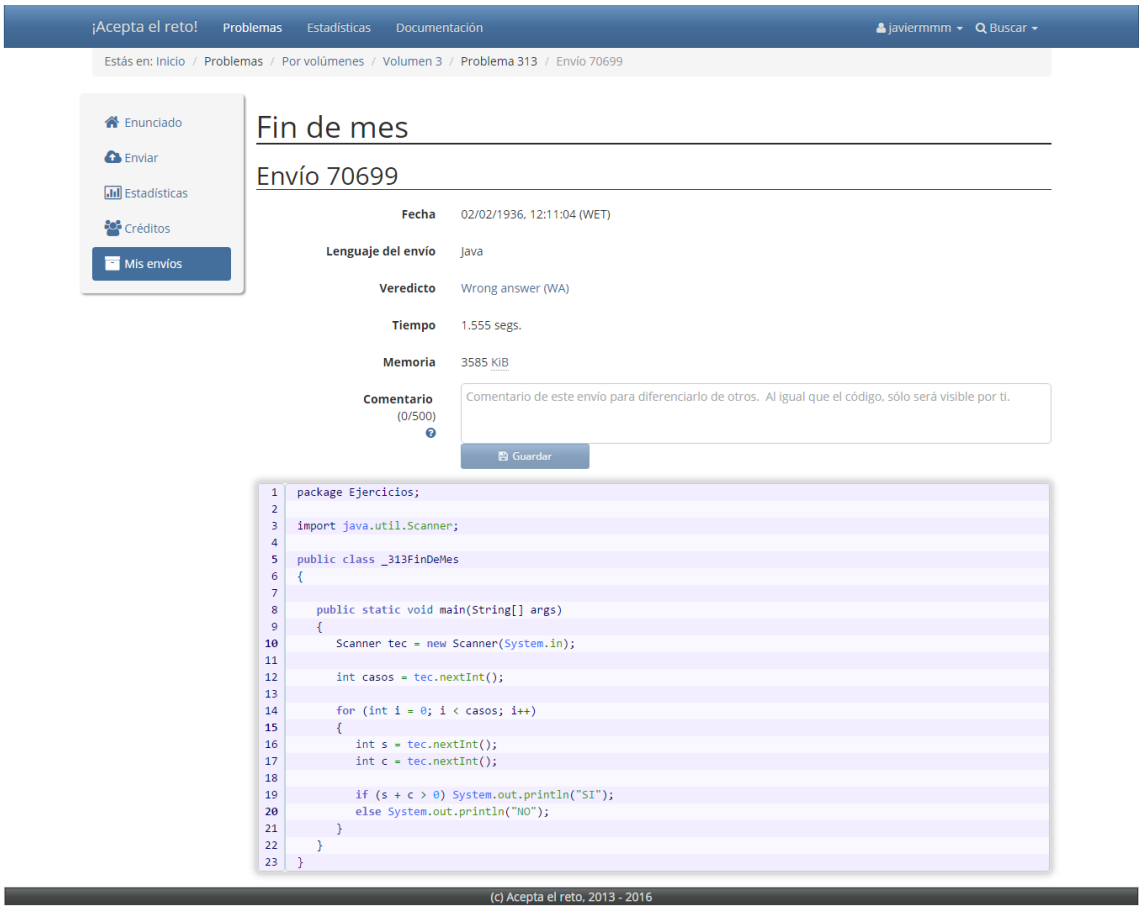
¡Acepta el reto!

¡Acepta el reto! [7] es una web lanzada desde la Facultad de Informática de la Universidad Complutense de Madrid, por un grupo de profesores del Grupo de Aplicaciones de Inteligencia Artificial (GAIA), hace tres años aproximadamente.

El origen del concepto del juez de Acepta el Reto, nace como consecuencia de la celebración del concurso de programación ProgramaMe que tiene lugar cada año, desde hace 4, en la facultad de informática de la Universidad Complutense de Madrid y que somete a equipos de toda España formados por tres alumnos y un profesor (como entrenador), a la resolución de 10 problemas de programación en un intervalo de 4 horas.

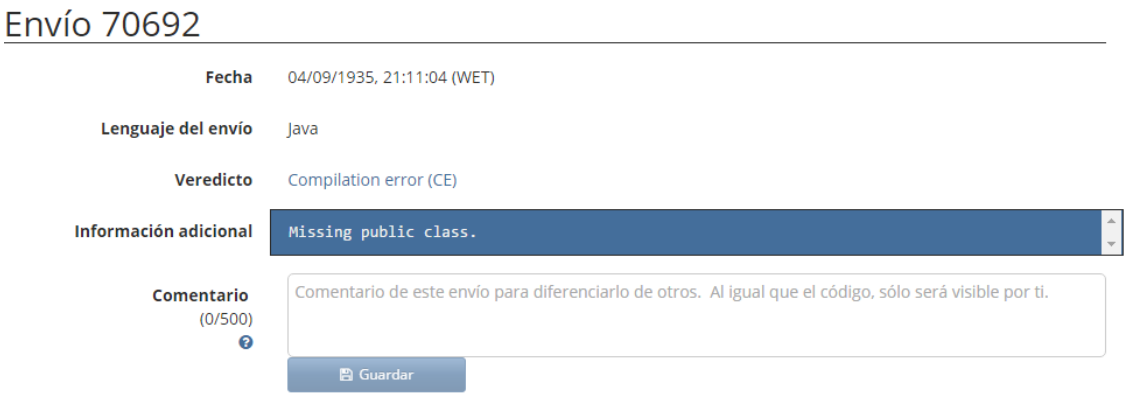
Además, ¡Acepta el reto! va creciendo con el tiempo y ya cuenta con más de 3.000 usuarios registrados, más de 200 problemas publicados, y unos 2.000 envíos al mes aproximadamente.

Este juez es sobre el que se pretende aplicar lo realizado en el presente trabajo de fin de máster y añadir así pistas que orienten a los usuarios sobre los envíos no aceptados por el juez de ¡Acepta el reto!, ya que en la actualidad, la única ayuda en este sentido que se obtiene directamente del juez, es el veredicto final del envío, aunque se permite adjuntar un comentario al envío, en caso de que queramos anotar alguna observación útil sobre el código del envío.



Acepta el reto 1

Tan sólo en los veredictos CE (Compilation Error), se muestra la salida del compilador como ayuda en el envío. Por lo que en caso de superar el límite de tiempo o memoria, o sacar una respuesta equívoca en un caso de prueba nos deja un tanto desamparados como usuarios.



Acepta el reto 2

Además, en cuanto a veredictos se refiere se dispone de los habituales y alguno un poco más concreto. El listado es el siguiente:

- AC (Accepted)
- PE (Presentation Error)
- WA (Wrong Answer)
- CE (Compilation Error)
- RTE (Runtime Error)
- TLE (Time Limit Exceeded)
- MLE (Memory Limit Exceeded
- OLE (Output Limit Exceeded)
- RF (Restricted Function)
- IE (Internal Error)

Sin embargo, en este caso existe una página en la documentación del juez, que explica con detalle cada veredicto, su significado y las posibles causas que lo pueden haber provocado, lo cual sirve de orientación, al recibir un veredicto negativo.

Punto de Partida

A continuación se exponen los componentes y elementos previamente existentes de los que se han hecho uso durante la realización del presente trabajo y que presentan cierta relevancia de cara a comprender mejor el funcionamiento de un juez evaluador de problemas.

Definición de los ejercicios

Al inicio del trabajo se disponía de un conjunto de herramientas, bautizadas con el nombre de ACREX, con las que operar de cara a la simulación de la evaluación de un juez. Entre estas herramientas se encuentran las clases utilizadas para cargar los problemas, los cuales pueden estar en un fichero comprimido de extensión .zip, o en un directorio.

Una vez tengamos el problema como un objeto que podemos manipular, podemos aprovechar los métodos que nos ofrece para acceder a su estructura y obtener los testcases[2] de los que disponga. En particular se identifican los denominados como empty y sample además de un conjunto de generadores que serán más exigentes con la solución del envío y que son creados por los autores del problema.

Estos generadores son los encargados de generar ficheros de entrada con casos de prueba para evaluar la solución enviada. Además, se dispone de una solución oficial para cada problema, que se ejecutará para obtener un fichero de salida con los resultados que se esperan conseguir en cada caso de prueba.

La denominación "solución oficial" se da a aquella solución que se considerará de referencia para que el juez genere los ficheros de salida. Al margen de resolver el problema, la solución de referencia también comprueba la entrada, lo que significa que los generadores de los casos de prueba son validados. La ejecución de esta solución sirve, además, como tiempo de referencia razonable a la hora de establecer el tiempo límite permitido en los envíos del problema para obtener un veredicto favorable.

Igualmente, por motivos de eficiencia, se pretende que durante la prueba del problema no sea necesario realizar una ejecución de la solución del usuario por cada caso de prueba, lo cual permite una evaluación más exhaustiva, por

[2] En ocasiones, la nomenclatura puede parecer confusa en cuanto a la diferencia entre un caso de prueba y un conjunto de ellos, por ello, a lo largo de este documento se utilizará 'testcase' para referirse a un fichero con un conjunto de casos de prueba, donde cada caso de prueba se representará con los datos de entrada que espera el problema.

lo que los problemas se diseñan con una entrada multicaso, para así evaluar varios casos con una sola ejecución.

No obstante, en esta circunstancia se dificulta fuertemente la apreciación de los casos concretos en los que se han producido los errores, puesto que todos serían ejecutados uno tras otro. Sólo en aquellos problemas en los que la salida consiste en una única línea por cada caso de prueba, resulta sencillo saber cuál es el primer caso que falla, pero dada la necesidad de ejecutar problemas que generan más de una línea de salida por caso, se debe buscar una solución más general.

Por esta razón se necesita un mecanismo que nos otorgue la capacidad de identificar dónde acaba un caso y dónde comienza el siguiente, permitiéndonos investigar los ficheros de salida (y de entrada) para poder sacar información adicional o procesar las salidas de la manera que nos convenga. Este mecanismo, no es otro que de la colocación de marcas en los puntos adecuados para etiquetar de esta forma los casos.

Por tanto, se debe destacar también que tanto los ficheros de entrada creados a partir de los generadores, como los ficheros de salida generados por las soluciones, pueden ser generados con y sin etiquetas. Estas etiquetas son marcas especiales que se introducen en el código del generador para marcar el inicio del conjunto de casos de prueba, el fin de cada caso de prueba y el fin del conjunto de pruebas.

Las etiquetas deberán colocarse con cuidado, pues aunque en el caso de la generación de salidas con etiquetas, solo se usará la marca de separación de casos, por no aplicarse los formatos de entrada, ésta última sí puede variar de un problema a otro. Concretamente, el uso de las marcas en un generador sería el siguiente:

- INICIO_CASOS: Debe aparecer justo antes del primer caso. En los problemas en los que el formato de entrada sea del tipo en el que se indica primero el número de casos que se deben procesar, la marca habrá de situarse después de escribir el número de casos. A este fragmento se le denomina prólogo.
- FIN_CASO: Debe ubicarse tras haberse escrito el contenido de cada caso. Actúa de separador de casos de prueba.
- FIN_CASOS: Debe situarse inmediatamente después de la marca FIN_CASO del último caso de prueba escrito. Indica el final de los casos de prueba y el comienzo de lo que se denomina epílogo. Este epílogo consiste en el caso especial que marca el fin de la entrada en caso de tener ésta, dicho formato, por lo que se ignorará todo lo que pudiera venir detrás.

Con este objetivo, se definen las tres respectivas constantes del precompilador inicialmente vacías, lo cual quiere decir que las aplicaciones generadas, no ven modificado su comportamiento y pueden generar los ficheros pertinentes (.in en caso de referirnos a un generador o .out en caso de referirnos a una solución) de forma correcta y "limpia", con la ventaja de que al compilar se pueden especificar ciertas opciones que permitan la impresión de esas marcas en los ficheros sustituyendo los símbolos por llamadas a printf que escriben los caracteres especiales a modo de indicación.

Estas indicaciones nos sirven para poder hacer algunas cosas interesantes de cara a la generación de pistas, como:

- Conocer el número de casos de prueba que hay en un testcase (fichero .in).
- Averiguar el esquema usado para definir la entrada (como se verá a continuación).
- Saber si los casos de prueba consisten en, y/o generan una única línea.
- Comprobar si un .out de una solución oficial responde a todos los casos de prueba (para detectar fallos en la solución oficial).
- Descubrir cuál es el primer caso que falla, en caso de haberlo, a partir de un fichero de salida.
- Generar nuevos ficheros de entrada o salida, a base de seccionar casos de prueba concretos o subconjuntos de ellos.

Así pues, la razón de utilizar C++ como lenguaje de programación para la implementación de los generadores y de la solución oficial, responde a la necesidad de la existencia de un precompilador que permita la definición de las mencionadas etiquetas, ya que en java, por ejemplo, no se dispone de esta funcionalidad.

Además, resulta necesario comentar los distintos esquemas que puede seguir la entrada de un problema:

- Se lee primero el número que indica cuántos casos van a continuación y después se lee ese número de casos. Ejemplo: *"La entrada comienza con un número que indica cuántos casos de prueba vendrán a continuación".*

```
4
100 -10
-10 -100
-10 100
100 -1000
```

- Se leen casos de prueba hasta que se llega a un caso especial que marca el final del testcase y que no se procesa. Ejemplo: *"La entrada termina con un caso en el que num < den, que no debe procesarse."*

```
5 2
6 5
8 3
17 8
1 2
```

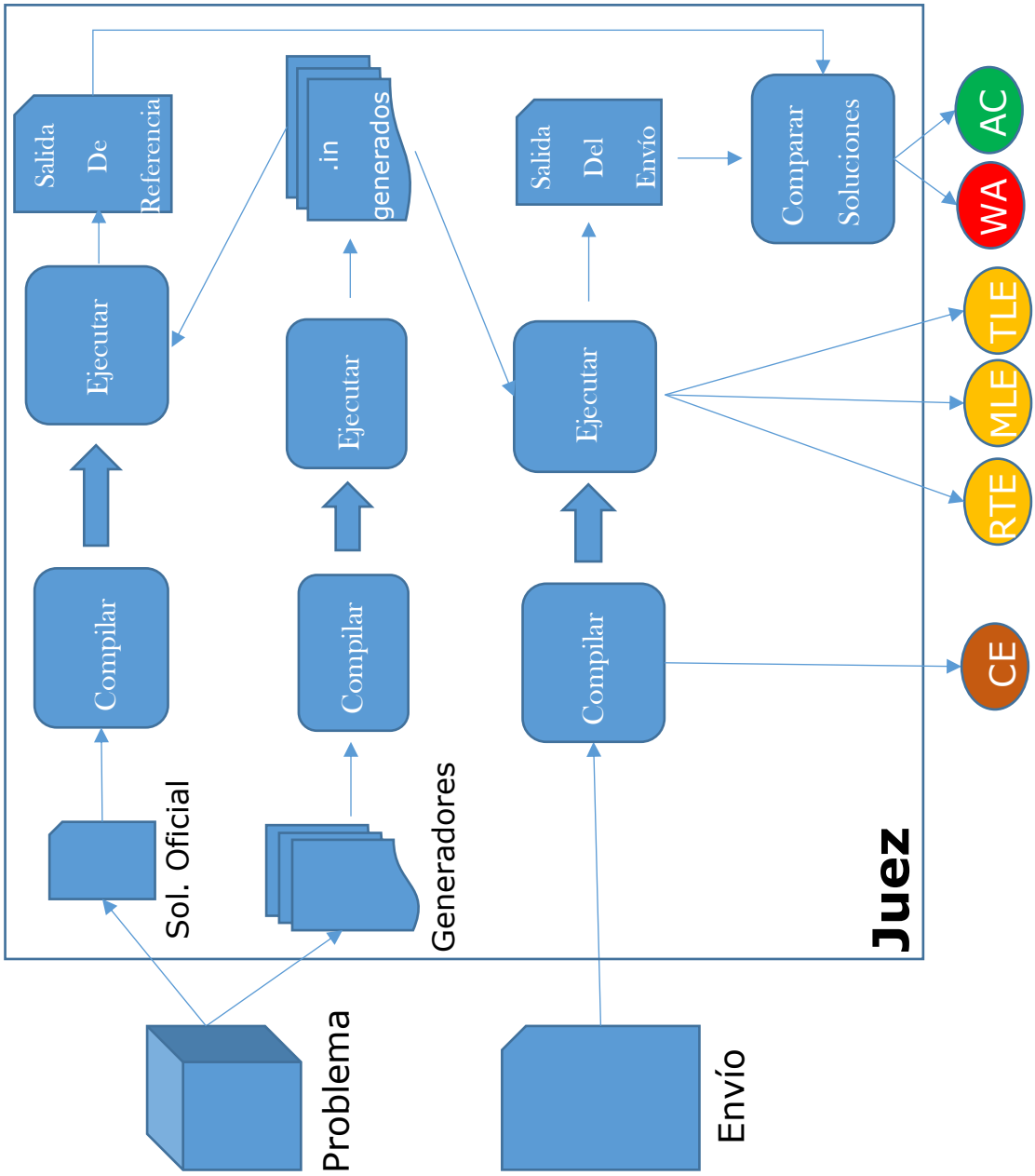
- Se leen los casos, uno por uno, hasta que se encuentre EOF (fin del fichero):

```
H.$.*M
H==@M
MHMHMHHMM
```

Asimismo, se dispone de las clases y métodos necesarios para compilar código, así como para ejecutar aplicaciones, que como cabía esperar son probablemente las más utilizadas. De hecho, usando la vía reduccionista, lo que se hace en realidad, es ejecutar procesos de distintos tipos: un .exe generado al compilar un envío en C++, el compilador de java (javac), al compilar un envío en este lenguaje, o g++ (por ejemplo), al compilar un envío implementado en C++.

A continuación se describe un posible esqueleto de generador contemplando los distintos formatos de entrada, la colocación de las etiquetas, así como también se marcan los fragmentos destinados a los mencionados prólogo y epílogo.

```cpp
#include <stdlib.h>
#include <iostream>
using namespace std;

// Include y definiciones para las marcas
#include <stdio.h>
#ifndef INICIO_CASOS
#define INICIO_CASOS
#define FIN_CASO
#define FIN_CASOS
#endif

int main() {
	// [Prólogo]
	// Número de casos de prueba (en caso de usar ese formato)
	// cout << "2" << endl;

	INICIO_CASOS;

	// Caso 1
	// ...
	FIN_CASO;
	// Caso 2
	// ...
	FIN_CASO;

	// Final
	FIN_CASOS;

	// [Epílogo]
	// Marca de fin de entrada (en caso de usar ese formato)
	cout << "-1" << endl;

	return 0;
}
```



Esquema de funcionamiento 1

HintExtractor

Entrando más en materia referente a información de los envíos, se dispone de una herramienta bautizada con el nombre de HintExtractor, que a partir de un fichero de entrada con y sin etiquetas, un fichero de salida con y sin etiquetas, la solución generada por el envío, y un objeto con algunas funciones de comparación, nos puede aportar información sobre el envío en caso de que la comparación resulte en que la solución generada no es igual que la solución de referencia.

El mencionado comparador, dispone de algunas funciones que dados un fichero de entrada y las salidas oficial y generada, compara estas dos últimas. El fichero de entrada en realidad resulta prescindible en la actualidad ya que no se utiliza, sin embargo, figura como parámetro de varios métodos de esta clase, pensando en posibles cambios, o ampliaciones de las comparaciones para un futuro. La comparación se realiza a nivel de byte, es decir, leyendo byte a byte ambas soluciones y comparándolas fijándonos también en que acabemos la lectura a la vez en ambas soluciones, dicho de otra forma, que ninguna de las dos tiene más (o menos) contenido del esperado.

La información más útil que podemos extraer de este proceso es el número de caso en el que se ha producido la discrepancia, la entrada que se leyó o la salida que se esperaba; información ésta que será utilizada como pista del envío en caso erróneo, como se comentará más adelante.

Por otra parte, esta herramienta dispone también de algunos métodos adicionales interesantes, como por ejemplo, un método que indica si no se puede devolver información relevante por no haber ningún caso erróneo, otro para obtener el número de casos totales del fichero, y otro para comprobar si el fallo es debido a que el contenido de la salida generada por el envío es correcto pero contiene demasiado texto, es decir, incluye contenido adicional que no debería aparecer.

Se debe tener en cuenta también que, en caso de existir más de un error en la comparación, la información devuelta es la del primer caso de error, además de que si se realiza más de una comparación utilizando esta herramienta, sólo se hace el trabajo la primera vez, y en las siguientes la operación no tendrá efecto.

Como cabe esperar, la información del error será de utilidad a la hora de construir una pista referente al envío del usuario, sin embargo, esta herramienta no devuelve la salida leída, resultado de la ejecución de la solución enviada, por lo que esta información, si se quiere, deberá obtenerse por otra vía.

Implementación de pistas

En este capítulo, se explicará el trabajo que se ha llevado a cabo durante la realización del proyecto que nos ocupa.

Toma de decisiones

Tras estudiar varios jueces con el fin de comprobar si disponían de pistas sobre los envíos de los usuarios, y en caso afirmativo, de qué forma se dan y con qué contenido, se decide dar las pistas en relación al veredicto obtenido del juez, sin embargo, el juez establece cierta jerarquía en cuanto a la prioridad o nivel de importancia de los veredictos que una solución obtiene.

Esta jerarquía se establece por el hecho de que al probar un envío se van a realizar varias ejecuciones, cada una de las cuales se probará con un conjunto de entradas distinto. Dicho de otra forma, se leerán las entradas de testcases distintos. En general, el primer caso que prueba es con la entrada vacía, es decir, se mantiene un fichero empty.in sin entradas escritas. El siguiente es el fichero sample.in, que tendrá como entradas las expuestas en el ejemplo que aparece en el enunciado, normalmente 3 ó 4 casos. Y por último, los denominados testcaseN.in, que incluirán muchos más casos de prueba y que se corresponderán con los generadores creados por los autores del problema. Estos testcases serán más exhaustivos, y a menudo, incrementales; esto quiere decir que los casos que se prueben en testcaseN1 será más probable que sean superados que los del testcaseN2, y así sucesivamente con el resto de testcases, aunque en algunos problemas el planteamiento puede ser distinto.

En ¡Acepta El Reto!, la prioridad de los veredictos más significativos vendría a ser de la forma más habitual, que es la siguiente:

CE > RTE > TLE > MLE > WA > PE > AC

Sin embargo, de cara a dar pistas sobre un envío fallido, la estrategia que se ha elegido es distinta. En primera instancia, se comprueba en qué testcase se producen los errores, de tal forma que la prioridad en ese caso sería algo como:

empty > sample > testcase1 > testcase2 > … > testcaseN

De esta forma, si en el testcase vacío se obtiene un resultado que no sea aceptado, será ese el resultado que se tendrá en cuenta para dar la pista. Si

el empty recibe un AC, se procederá de manera análoga con el sample y así sucesivamente con el resto de testcases.

Como se aprecia por el orden de precedencia anterior, en igualdad de prioridad se dará pista para el primer error ocurrido y en cuanto a los veredictos, se aplicará después la prioridad establecida por el juez.

En los dos primeros testcases, sin embargo, dada la simplicidad de los casos, la pista dada será bastante escueta, cuanto menos, y se limitará a una breve frase indicando el fichero de casos de prueba que ha fallado. Algo del estilo: *"Error en el fichero empty/sample"*. Esta decisión se toma por considerar estos ficheros de casos de prueba como fácilmente superables, además de que la información de las entradas que contienen, se encuentra al alcance del usuario desde el principio, la del empty por estar vacío, y la del sample por aparecer en el enunciado.

Por otra parte, para aquellas soluciones que pudieran ser más complejas, se debe tener en cuenta que al establecer una prioridad en los veredictos por encima de la del propio juez, puede darse un resultado que no sea equivalente con el veredicto oficial. Por ejemplo, supongamos un envío, con los siguientes resultados en sus ejecuciones:

FICHERO	VEREDICTO
Empty	WA
Sample	WA
Testcase1	TLE
Testcase2	TLE

En este envío, el juez devolvería un veredicto de TLE, por tener mayor prioridad, sin embargo, la pista que se daría sería del estilo de *"Error en el fichero empty",* por anteponer la prioridad de los testcases a la de los veredictos. En un primer momento, esto puede parecer incongruente, pero lejos de constituir un error, se aporta un valor adicional, ya que si de partida no se supera el test del fichero de prueba vacío, difícilmente se va a alcanzar una solución aceptada que cubra todos los casos posibles, además de que el juez únicamente devuelve un veredicto, por lo que resulta complicado averiguar en cuál de los ficheros de casos de prueba se obtiene dicho veredicto y de esta forma, al menos se orienta para el fichero vacío y el de ejemplo.

Tipos de pistas

Para el resto de testcases, esto es, los generadores preparados por el autor del problema, se plantean pistas algo más detalladas. Inclusive, se comprueba si el error detectado es debido a una posible dependencia entre casos. Cuando probamos la solución, para evitar hacer una ejecución por cada entrada que queremos probar, lo que se hace es meter la solución dentro de un bucle para aplicarla un número de veces que dependerá del generador implementado, por lo que es posible que debido a la forma en que esté programada la solución enviada, se den situaciones en las que de un caso a otro se arrastre algún valor o comportamiento que, por acumulación, provoque el fallo en algún caso posterior. En estas situaciones concretas, lo que se hace es generar nuevos ficheros de entrada y salida que contendrán únicamente la entrada y salida, respectivamente, del caso que provocó el error, para volver a ejecutar después, esta vez aisladamente, el caso defectuoso. Tras comprobar si el error persiste, se informará al usuario oportunamente.

A continuación se muestra una lista de las categorías de pistas identificadas:

1. EMPTY
2. SAMPLE
3. RTE CON DEPENDENCIA
4. RTE SIN DEPENDENCIA
5. TLE CON MÁS SALIDA ESCRITA PERO BUENOS RESULTADOS
6. TLE CON MÁS SALIDA ESCRITA Y MALOS RESULTADOS
7. TLE CON BUENOS RESULTADOS
8. TLE CON MALOS RESULTADOS
9. WA CON DEPENDENCIA
10. WA SIN DEPENDENCIA
11. WA POR SALIDA GENERADA MÁS LARGA

Como puede apreciarse, no se generan pistas para los veredictos MLE, pero es debido a que es necesario hacer uso de la infraestructura para tal efecto de la que dispone el juez, y no resulta sencillo en absoluto, motivo por el que no se ha considerado su inclusión. No obstante, la herramienta es fácilmente escalable para añadir el soporte al veredicto de límite de memoria superado, ya que la estrategia a seguir será muy parecida a la seguida en el caso de los TLE. Aparte de esto, el veredicto MLE, no resulta especialmente importante, ya que en muchos jueces ni siquiera existe, y tampoco suele ser empleado

en concursos de programación, por ello, su inclusión sólo repercutiría a la hora de integrarlo en el juez de Acepta el Reto.

Pistas finales

Teniendo en cuenta las categorías identificadas, lo primero es montar un juez a pequeña escala que nos permita reproducir el envío de ficheros, para ello, construiremos una aplicación de consola, que dados un problema y una solución, nos devuelva el resultado de la evaluación de sus testcases y una pista en caso de haber encontrado algún problema.

Lo primero será inspeccionar los datos de entrada de nuestra aplicación, ya que necesitamos cargar el problema para tener una representación java del objeto que podamos manipular, y averiguar el lenguaje de la solución que se va a probar. Conociendo este último dato, procederemos a buscar en la máquina el compilador necesario para compilar después la solución del usuario.

Asimismo generaremos todos los ficheros de entrada y de salida tanto con etiquetas como sin ellas.

Una vez hecho esto, comenzaremos a ejecutar la solución, utilizando cada vez las entradas necesarias. El primer fichero de entrada que usaremos será el empty.in y tras su ejecución comprobaremos si ha tenido lugar algún error, de manera que si lo hubiera, cualquiera que fuese el error o el veredicto arrojado, se devolverá la misa pista en todos ellos:

ERROR en la prueba con el fichero empty.

Después es el turno de los casos de ejemplo del enunciado contenidos en el fichero sample.in, con el que se procederá de manera análoga, y de nuevo se devolverá la misma pista en caso de haber error, que independientemente del veredicto será como esta:

ERROR en la prueba con el fichero sample.

Una vez pasadas estas pruebas, comenzaremos a probar con todos y cada uno de los ficheros testcases generados, pero esta vez usaremos distintas estrategias dependiendo del tipo de veredicto que se devuelva y de la salida que se haya generado al ejecutar la solución probada con la entrada del testcase.

Seguidamente se exponen dichas estrategias en base al veredicto en el que se aplican.

RunTimeError

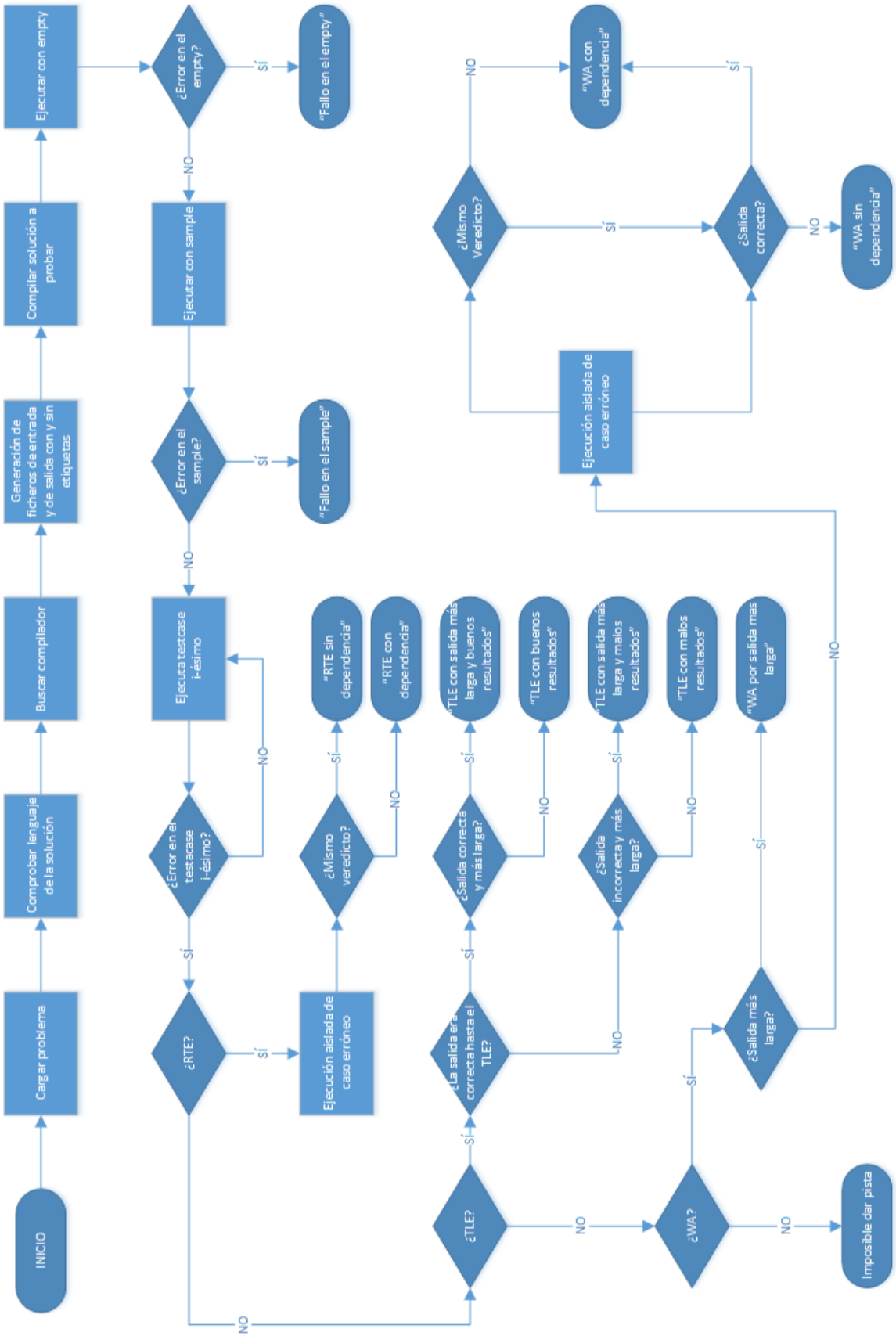
Si obtenemos un error de ejecución con un caso de prueba, se preparará una ejecución aislada de dicho caso. Para ello, utilizamos la herramienta HintExtractor, para conocer cuál es el caso en el que nos hemos quedado. Lo recortamos y nos lo guardamos junto con la respectiva salida que tomamos de la solución de referencia. Al margen de esto, como ya hemos comentado previamente, un problema puede tener tres tipos distintos de entrada, por lo que le preguntaremos al testcase también por dicho formato. Así, dependiendo del formato obtenido construiremos un fichero de entrada de una u otra forma, para copiar en él la entrada que recortamos previamente y efectuar la ejecución aislada del caso de prueba.

- Tras la re-ejecución, nos fijaremos en la salida obtenida. Si ese caso suelto, obtiene de nuevo RuntimeError, se devuelve una pista indicando:

ERROR en tiempo de ejecución en el caso <caseNumber> con entrada <inputRead>

- Si se obtiene un resultado válido (Accepted), o si se superara el tiempo límite establecido (o en un futuro, la memoria), se indica una posible dependencia entre casos:

ERROR en tiempo de ejecucion en el caso <caseNumber> con entrada <inputRead>
Tras la re-evaluación del caso de prueba aislado, no se ha obtenido veredicto RTE.
CONCLUSIÓN: Existe una dependencia de casos en tu solución



Flujo de generación de pistas 1

TimeLimitExceeded

Para las ejecuciones que obtienen un veredicto de tiempo límite superado, se lleva a cabo una comparación entre la salida que se ha generado hasta el momento de superar el umbral de tiempo permitido y la solución de referencia, para comprobar ciertas condiciones y devolver una pista u otra según el caso. Las condiciones son las siguientes:

- Que la solución generada no sea vacía, y además sea prefijo de la solución de referencia. En este caso la pista que se devuelve es:

Te pasaste del tiempo límite, pero hasta este punto ibas bien con las salidas generadas

---

- Si la salida generada por la solución no es prefijo de la salida de referencia, puede ser por tener algún caso erróneo, en cuyo caso se devuelve la pista:

Te pasaste del tiempo límite, pero las salidas que generabas eran incorrectas

---

- La solución generada puede no ser prefijo de la solución oficial, por escribir de más. En esta situación, se comprueba además si los resultados iban bien o no, comprobando en esta ocasión si la solución oficial es prefijo de la solución generada. Para los casos en los que lo es, y en los que no, se dan, respectivamente, las siguientes pistas:

Te pasaste del tiempo límite y escribes de más, pero tus respuestas iban bien

---

Te pasaste del tiempo límite y escribes de más, pero además tus respuestas eran incorrectas

---

WrongAnswer

Para identificar un veredicto de respuesta incorrecta, tras la ejecución de la solución se comparan los resultados obtenidos con los resultados de la solución de referencia, y en este caso, también es posible detectar distintas situaciones que harán que la pista devuelta cambie. Una vez comprobado que los resultados de la solución de referencia y de la solución generada a partir del envío, son distintas, comprobamos las condiciones que se exponen a continuación.

- Que la solución generada sea más larga que la solución de referencia devolverá la siguiente pista:

La longitud de la salida generada, es mayor de la esperada.
Quizá estés iterando más de lo debido.

En caso de que el tamaño sea igual, el caso se reduce a la existencia de un caso erróneo así que utilizando la herramienta HintExtractor, identificamos el caso que falla y lo aislamos para re-ejecutarlo y comprobar una posible dependencia. De nuevo, de manera análoga al caso de RTE, será necesario comprobar el formato de la entrada del problema y construir el fichero de entrada que contendrá únicamente el caso que falla de la forma apropiada. Tras esto se realiza la segunda ejecución y llegados a este punto se han distinguido tres situaciones posibles:

- Si el resultado vuelve a dar error, se devuelve una pista de este estilo:

Error en el caso <caseNumber>
Entrada leída = <inputRead>
Salida esperada = <outputExpected>

- Si el resultado en esta ocasión es correcto, se devuelve lo siguiente:

Error en el caso <caseNumber>
Entrada leida = <inputRead>
Salida esperada = <outputExpected>
La evaluación aislada del caso de prueba ha obtenido una salida correcta.
CONCLUSIÓN: Tu código tiene dependencias entre casos de prueba

- Si el resultado devuelve otro tipo de problema (error de ejecución, tiempo superado, etc.), se devueve algo como esto:

Error en el caso <caseNumber>
Entrada leida = <inputRead>
Salida esperada = <outputExpected>
Durante la ejecución aislada del caso erróneo, se obtuvo un veredicto distinto y NO aceptado
CONCLUSIÓN: Quizá existe una dependencia de casos en tu solución

Evaluación y estadísticas

Tras la implementación descrita en el capítulo anterior, se ha realizado una evaluación usando la herramienta desarrollada para comprobar su eficacia, extraer algunas estadísticas y tratar de analizarlas. Se han evaluado alrededor de 3.500 envíos reales, de 4 problemas distintos publicados en ¡Acepta el reto!, resueltos tanto en java como en C++[3].

Análisis de los resultados

Los problemas elegidos son los numerados en ¡Acepta el reto! con el 100, el 116, el 140, y el 313. Los envíos se corresponden con alrededor de 1000 por cada uno de los tres primeros y unos 500 pertenecientes al último de ellos.

Tras las pruebas realizadas, vemos cómo algunas de las pistas son más comunes que el resto, como son las relativas a los errores producidos en los ficheros de prueba más sencillos, el empty y el sample.

Una vez superadas estas pruebas, los errores más comunes son los que en principio cabía esperar, es decir, errores en tiempo de ejecución sin dependencia de casos, y salidas incorrectas también sin dependencia, no obstante merece mención especial, el caso de los errores por superar el límite de tiempo establecido, pues a la luz de estos resultados, en la mayor parte de estas evaluaciones, las salidas generadas hasta el momento del TLE, contenían errores.

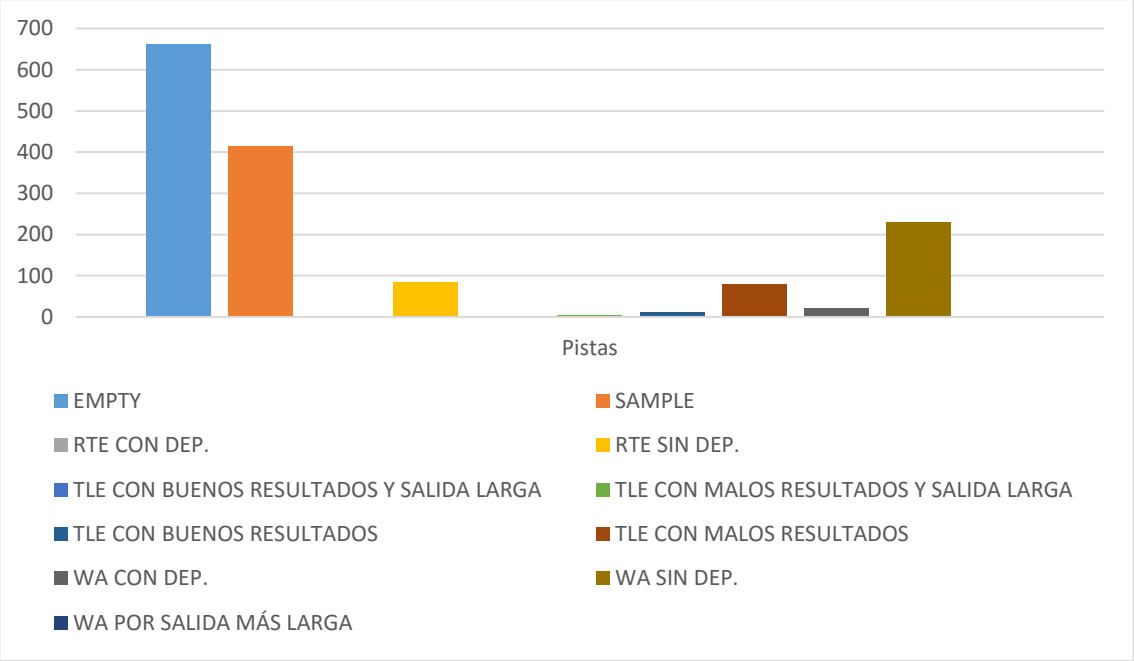
En la tabla no figura ninguna pista para el veredicto de MLE (Memory Limit Exceeded), pues como ya se ha mencionado, no se utiliza la infraestructura necesaria para ello. Por este motivo, los veredictos que el juez originalmente detecta que superan el límite de memoria establecido, son evaluados con otro veredicto por la herramienta desarrollada, por lo que los MLE se encuentran repartidos entre el resto de veredictos soportados, incluyendo el AC (Accepted). Sin embargo, de entre los cerca de 3500 envíos evaluados de los cuatro problemas presentados, sólo 9 fueron evaluados por el juez con MLE, de los cuales solamente 1 fue considerado AC.

Esto sucede porque como el mecanismo de ejecución utilizado es distinto del que usa el juez, los envíos analizados son de cuatro problemas cuyas restricciones de memoria no son especialmente fuertes, para así poder minimizar el número de veredictos de este tipo.

[3] El juez admite soluciones también en C, pero el uso de este lenguaje en el juez es minoritario, y aunque el extractor funciona igual, dada la poca cantidad de envíos en este lenguaje, resulta más complicado encontrar ejemplos significativos.

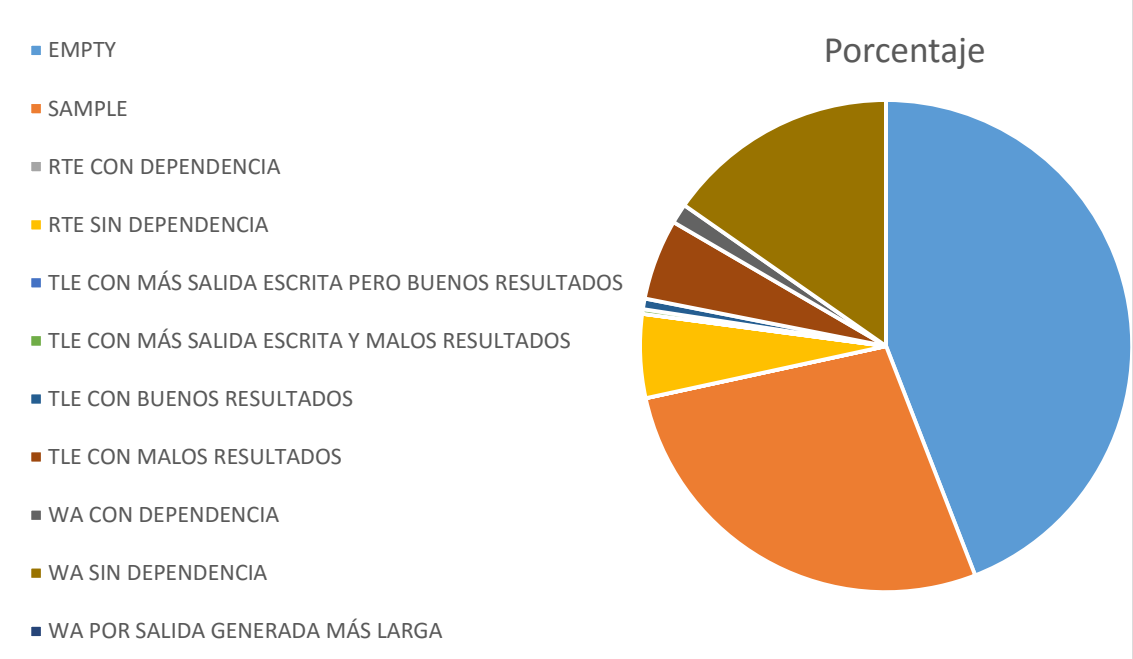
A pesar de ello, el envío que se considera aceptado demuestra que sólo esa solución era MLE y únicamente MLE, y que el resto de envíos contenía errores adicionales que la herramienta desarrollada detecta y para los cuales proporciona una pista acorde.

Tipo de Pista	Nº
EMPTY	661
SAMPLE	413
RTE CON DEPENDENCIA	0
RTE SIN DEPENDENCIA	83
TLE CON MÁS SALIDA ESCRITA PERO BUENOS RESULTADOS	0
TLE CON MÁS SALIDA ESCRITA Y MALOS RESULTADOS	4
TLE CON BUENOS RESULTADOS	10
TLE CON MALOS RESULTADOS	79
WA CON DEPENDENCIA	20
WA SIN DEPENDENCIA	230
WA POR SALIDA GENERADA MÁS LARGA	0
TOTAL	1500



Número de pistas de cada tipo 1

Tipo de Pista	%
EMPTY	44,1%
SAMPLE	27,5%
RTE CON DEPENDENCIA	0%
RTE SIN DEPENDENCIA	5,5%
TLE CON MÁS SALIDA ESCRITA PERO BUENOS RESULTADOS	0%
TLE CON MÁS SALIDA ESCRITA Y MALOS RESULTADOS	0,3%
TLE CON BUENOS RESULTADOS	0,7%
TLE CON MALOS RESULTADOS	5,3%
WA CON DEPENDENCIA	1,3%
WA SIN DEPENDENCIA	15,3%
WA POR SALIDA GENERADA MÁS LARGA	0%



Porcentaje de pistas de cada tipo 1

Ejemplos

Para ilustrar qué tipo de errores son los responsables de las pistas que se generan, se exponen a continuación alguno de los envíos probados y sus respectivos veredictos y pistas.

Fallo en el empty

Para ejemplificar este caso, usaremos un envío en C++ para resolver el problema número 100, llamado "Constante de Kaprekar"[4], disponible en https://www.aceptaelreto.com/problem/statement.php?id=100.

En el enunciado se pide, dado un número de cuatro cifras, aplicar la rutina de Kaprekar, hasta obtener la constante de Kaprekar, es decir el número 6174. Se debe devolver el número de iteraciones necesarias en la rutina para lograrlo, sabiendo que el número máximo será en cualquier caso de 7. En el enunciado se explica en qué consiste la rutina de kaprekar, que básicamente debe reordenar los dígitos del número leído para formar dos nuevos números, uno con los dígitos ordenados de manera ascendente y otro, con los dígitos ordenados de manera descendente, para después restar el menor al mayor. El enunciado presenta varios ejemplos de dicha rutina, así como ejemplos de la entrada que se espera recibir, y de la salida que se debe generar.

A continuación se muestra una solución perteneciente a un envío real de un usuario de ¡Acepta el Reto!

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main () {
  int pos1, pos2, pos3, pos4, cont=0;

  //Se lee el caso
  cin >>pos1;

  while(pos1!=6174) {

      //Aquí se trata el número para ir guardando cada dígito suelto
      pos4=pos1%10;
      pos1=pos1/10;
      pos3=pos1%10;
      pos1=pos1/10;
      pos2=pos1%10;
```

[4] Enunciado ilustrado en la página 7 de este documento.

```
	pos1=pos1/10;

	//Se intenta ordenar las cifras
	if (pos1>pos4) {
		pos1 = (pos1*1000+pos2*100+pos3*10+pos4) -
		       (pos1+pos2*10+pos3*100+pos4*1000);
	}
	else
		if (pos1==pos4 && pos2>pos3) {
			pos1 = (pos1*1000+pos2*100+pos3*10+pos4) -
			       (pos1+pos2*10+pos3*100+pos4*1000);
		}
		else
			if (pos1==pos4 && pos2==pos3) {
				cont = 7;
				pos1=6174;
			}
			else {
				pos1 = (pos1+pos2*10+pos3*100+pos4*1000) -
				       (pos1*1000+pos2*100+pos3*10+pos4);
			}

		//se actualiza el número de iteraciones que se han hecho
		cont++;
	}

	//Se escribe el resultado
	cout << cont << endl;

	return 0;
}
```

Este envío fue evaluado en su momento por el juez con un TLE. Si analizamos el código podemos entender que el límite de tiempo se superará debido a que la ordenación de las cifras para realizar la resta, no se realiza adecuadamente. Sin embargo, tras someter este envío a la herramienta de extracción de pistas, se genera este resultado:

```
Resultado final de las evaluaciones:
empty ---> WA
sample ---> TLE
testcase1 ---> TLE
**************************************
Veredicto Final = WA

CATEGORIA: 1 - EMPTY
ERROR en la prueba con el fichero empty.
```

La pista que se da, arroja algo más de luz sobre el error existente, resultando en realidad, que ya desde la prueba más temprana el envío está fallando. Pero ¿por qué se obtiene un Wrong Answer para la prueba con el fichero empty?

De nuevo, fijándonos en el código, podemos ver que lo primero que se hace, tras declarar las variables necesarias, es leer el primer caso, lo cual supone el error en sí. Como hemos mencionado antes, en ¡Acepta el reto! los problemas pueden esperar tres tipos de entrada distintos, con el número de casos a probar al inicio, con un caso especial al final o con el fin de fichero después de leer todas las entradas. En este envío, lo que está pasando es que su autor plantea la solución para probarla una única vez, pero en el enunciado se especifica que al principio de todo debe leerse el número de casos que se van a probar. Por ello, cuando se intenta leer del empty.in, se está leyendo el 0 que indica que hay 0 casos que probar, y todas las divisiones y módulos resultan en 0, lo cual provoca que el código entre por la rama que considera que se ha llegado al final, y devuelve un resultado incorrecto, en este caso, un 8, cuando en realidad no se esperaba ninguna salida para ese caso.

Fallo en el sample

En este caso se va a ejemplificar un caso de error al evaluar el fichero sample. Para ello se toma la solución de un envío real para el problema número 140 de ¡Acepta el reto! llamado "Suma de dígitos", disponible en https://www.aceptaelreto.com/problem/statement.php?id=140.

En este problema se pide que dado un número entero y no negativo, se sumen sus dígitos y se escriba como salida dicho número precedido de las sumas necesarias, es decir, para el número 3433 escribir como salida 3 + 4 + 3 + 3 = 13. El problema explica que esta operación se debe hacer hasta que se encuentre un número negativo como entrada, y de nuevo presenta varios ejemplos.

La siguiente es una solución en C++ perteneciente a un envío real del mencionado problema de ¡Acepta el reto!

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

void walter (int num, int & gs) {
      //Si el número que leemos es menor que 10,
        no se descompone y se muestra tal cual
      if (num / 10 == 0) {
```

```
            gs += num;
            cout << num;
      }
      //Si es mayor, se le quita el último digito y se llama
        recursivamente con el resto del número
      else{
            gs += num % 10;
            walter (num/10, gs);
            cout << " + " << num % 10;
      }
}

int main () {
      int num, gs = 0;
      cin >> num;
      //Se opera mientras el número que se lea no sea el -1 (o menor)
      while (num >= 0) {
            walter (num, gs);
            cout << " = " << gs << '\n';
            cin >> num;
      }
      return 0;
}
```

En este caso, el envío fue evaluado por el juez con un WA, y debido a que la solución ofrecida no coincide con la esperada, la herramienta desarrollada, devuelve el mismo veredicto.

```
Resultado final de las evaluaciones:
empty ---> AC
sample ---> WA
testcase1 ---> WA
testcase2 ---> WA
****************************************
Veredicto Final = WA

CATEGORIA: 2 - SAMPLE
ERROR en la prueba con el fichero sample.
```

En este caso el veredicto coincide, pero vamos a fijarnos en el código, para comprobar si de verdad esto es correcto.

El usuario propone una solución recursiva, en la que se va dividiendo el número leído entre 10 para ir acumulando en la pila de llamadas los dígitos en orden inverso y así tenerlos en orden a la vuelta de la recursión. Sin embargo, la variable que hace las veces de acumulador, y va sumando los

resultados parciales para mostrar después el resultado, acumula también de caso a caso, y pese a que la descomposición de las sumas se haga bien y se escriba correctamente la salida, el acumulador no es debidamente re-inicializado entre caso y caso, dando así el veredicto de Wrong Answer. Además como esto es un defecto del código que afecta a todos los testcases, menos al vacío, el primer fallo se producirá, según el orden de evaluación en la prueba con el fichero sample, y de ahí la pista generada.

RTE sin dependencia

En este caso, vamos a revisar una solución de un envío realizado por otro usuario real para resolver, esta vez usando java como lenguaje de implementación, el problema 313 llamado "Fin de Mes", disponible en https://www.aceptaelreto.com/problem/statement.php?id=313.

El problema pide que dados dos números comprendidos entre -10.000 y 10.000 que representen el saldo en una cuenta bancaria y el cambio estimado en ella (entendido como ingresos menos gastos), se indique si se puede llegar a fin de mes o no escribiendo en la salida "SI" o "NO". De nuevo, el problema cuenta con una serie de ejemplos tanto para la entrada como para la salida, que reflejan además el tipo de entrada esperada, que en este caso es indicando al inicio el número de casos a probar.

Una vez puesto en contexto, vamos a ver una solución de un envío realizado por un usuario real de ¡Acepta el reto!

```
import java.util.Scanner;

public class Exe313 {

	public static void main (String[] args) {

		Scanner sc1 = new Scanner (System.in);

		//Se lee el número de casos
		short cas = sc1.nextShort(), res;

		while (cas!=0) {
			//Se leen los datos de entrada
			res = sc1.nextShort();
			res += sc1.nextShort();

			//Si la suma es mayor que cero, se llega a fin de mes
			if (res>=0) {
				System.out.println("SI");
			}
			else {
```

```
                    System.out.println("NO");
                }

                //Se actualiza el contador de casos
                cas--;
            }
        }
}
```

En este caso, el veredicto del juez fue de RTE, y la evaluación llevada a cabo por la herramienta de extracción de pistas de un envío, evaluó también con fallo en tiempo de ejecución la evaluación probada.

```
Resultado final de las evaluaciones:
empty ---> AC
sample ---> AC
testcase1 ---> RTE
testcase2 ---> AC
****************************************
Veredicto Final = RTE

CATEGORIA: 4 - RTE SIN DEPENDENCIA
ERROR en tiempo de ejecución en el caso 1 con entrada -100 -100
```

En este caso, las evaluaciones coinciden, pero vamos a analizar de nuevo si esto es correcto o no.

El código enviado es bastante sencillo y escueto y vemos cómo se hace una lectura inicial para el número de casos, y se itera después tantas veces como casos leídos, haciendo en cada iteración la lectura de los datos de entrada y la suma necesaria para saber si se llegará a fin de mes o no. La variable contador del bucle que indica la condición de parada del mismo se actualiza adecuadamente en cada iteración y parece que todo funciona correctamente, pero entonces ¿qué está fallando?

El resultado de la extracción indica que todas las pruebas van bien, salvo el testcase1, y que en esta prueba, además, el error se produce desde el principio ya en el caso 1. Si se produce al inicio, sólo puede ser por dos razones, o se produce un fallo en la lectura o evaluación de ese caso concreto, que como hemos visto, no parece un motivo plausible, o el error se produce incluso antes de eso, es decir en la lectura del número de casos a probar, que es donde efectivamente está fallando este código.

Este usuario asumió, que los autores del problema no pondrían en un solo testcase más de 32,767 casos de prueba, que es el rango positivo hasta donde alcanzan las variables de tipo short en Java, y por ello declaró la variable que lee el número de casos de este tipo. No obstante, como se ha

comentado a lo largo del presente documento, los testcases más elaborados prueban el envío de una forma más exhaustiva y en concreto este testcase1, que es el fichero que falla, contiene 40401 casos, por lo que al hacer la lectura, el valor se sale del rango soportado y se produce un error en tiempo de ejecución.

WA sin dependencia

Para este tipo de pista, vamos a ver una solución en C++ de un envío para el problema 116 de ¡Acepta el reto!, llamado "¡Hola mundo!", disponible en https://www.aceptaelreto.com/problem/statement.php?id=116.

El enunciado propone, que dado un número comprendido entre 0 y 5, ambos inclusive, se muestren tantas líneas con la frase "Hola mundo." como indique ese número. De nuevo, el enunciado pone un ejemplo de entrada y salida para mostrar la forma correcta de escribir el mensaje.

Con el enunciado en mente, vamos a ver una solución en C++ de un usuario real.

```
#include <iostream>
using namespace std;

int main () {
	int n;

	//Se lee el número de vueltas
	cin >> n;

	//Se itera para saludar.
	if (n > 0 && n < 5) {
		for (int i = 0; i < n; i++) {
			cout << "Hola mundo." << endl;
		}
	}

	return 0;
}
```

En este caso tanto el veredicto del juez, como el ofrecido por el extractor de pistas coinciden de nuevo, ya que la salida es incorrecta al compararla con la solución esperada.

```
Resultado final de las evaluaciones:
empty ---> AC
sample ---> AC
```

```
testcase1 ---> AC
testcase2 ---> AC
testcase3 ---> AC
testcase4 ---> WA
****************************************
Veredicto Final = WA


CATEGORIA: 10 - WA SIN DEPENDENCIA
Error en el caso 1
Entrada leida = 5
Salida esperada =
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
```

Para este sencillo ejemplo, vemos que el código presentado supera todos los testcases salvo el último, en el que se propone como entrada el número 5.

A la vista del código se puede apreciar que tanto el bucle, como el if que lo envuelve excluyen el caso con entrada igual a cinco, por lo que en ese caso, el código no genera ninguna salida. No obstante, en estos casos se indica en la pista lo que se debería devolver para la entrada que se ha leído, por lo que la pista es realmente útil a la hora de corregir una salida errónea.

Conclusiones

A continuación, se plantean algunas de las conclusiones extraídas durante la elaboración del presente proyecto tanto a nivel personal, como en cuanto a recogida y análisis de ciertas estadísticas y se finalizará con una exposición de aquellos objetivos que se dejan planteados de cara al futuro, por no haberse podido alcanzar en esta ocasión.

Valoración personal

En primer lugar, querría expresar la cualidad continuista del presente proyecto, pues supone una extensión a mi Trabajo de Fin de Grado, en cuanto a funcionalidad de un juez online de programación. Sin embargo, a diferencia de aquella ocasión, esta vez he cambiado de lado en el desarrollo del proyecto, para estar más pegado al juez evaluador.

En cuanto al proyecto, se ha logrado cumplir todos los objetivos básicos que se habían propuesto al inicio, respetando los requisitos existentes.

En un primer momento, se partió de un conjunto de aplicaciones y herramientas ya implementadas con las que tuve que familiarizarme, lo cual dada la cantidad de código existente, constituyó el primero de los retos afrontados, aunque afortunadamente pude superarlo y continuar con el trabajo planteado.

Con este proyecto, se provee al profesor, y en un futuro al propio juez, de una herramienta de evaluación de envíos capaz de dar pistas útiles a soluciones enviadas con errores de distinto tipo.

La detección de errores por parte de un profesor de código que no es suyo, puede no ser siempre trivial dependiendo de la implementación que se esté sometiendo a la evaluación. Sobre todo para los casos con dependencia, es posible que por la forma en que se programó la solución, el código oculte esta dependencia y cueste un esfuerzo adicional, el hecho de hallar la causa del problema, sin embargo, automatizando el proceso, se reduce el tiempo empleado en ello.

Además, esta funcionalidad es una de las mayores carencias del juez (y una de las más demandadas por parte de los usuarios), por lo que la inclusión de esta funcionalidad se puede asumir como un factor diferenciador con respecto a otros jueces, pues si se echa la vista atrás, podemos recordar que el aspecto de las pistas en los envíos constituye un punto flaco en muchos de los jueces.

Asimismo, quisiera comentar la estrecha relación entre los códigos de las soluciones de problemas probadas y el comportamiento de la aplicación, pues en mi caso han supuesto el descubrimiento de una gran cantidad de técnicas

de programación, que bien no había visto antes, bien no son frecuentes ver o bien utilizan un enfoque que resulta, cuanto menos, curioso. Y es que han sido varias las ocasiones en las que gracias a soluciones de este tipo, me he acabado dando cuenta de que en ciertas situaciones límite muy concretas, mi código no cubría todos los casos posibles, lo cual, claramente, ha supuesto otro de los retos que me he visto en la tesitura de enfrentar, y por suerte, superar.

También existe otra conclusión que podemos extraer, atendiendo a las estadísticas recogidas y el análisis acometido, y es que las pistas que se dan, realmente ayudarán a los usuarios tal y como se ha ejemplificado en el capítulo anterior.

Así, aunque es cierto que en determinadas pistas la información podría ampliarse, la razón de no hacerlo no es otra que la de mantener la filosofía del juez sobre el que trabajamos, es decir, mantener cierto nivel de dificultad para que pueda suponer un desafío, pero teniendo al menos una pista para saber por qué dirección continuar. Esta característica supone, en mi opinión una ventaja doble, pues aparte de la ayuda que ofreces al usuario, eso repercute de igual forma en su tasa de éxito y en última instancia también en su moral, que aumentará al mismo ritmo que su utilización del juez, la cual poco a poco irá encadenando desafíos, "fidelizando" así al usuario.

Por ello, considero que esta herramienta será de gran utilidad y aportará grandes beneficios, sobre todo en el ámbito docente/pedagógico, pues el uso de las pistas será generalizado entre alumnos, profesores y los distintos participantes de concursos de programación.

Trabajo Futuro

Lamentablemente, pese al tiempo y el esfuerzo dedicado en la realización de este proyecto, hay ciertas características que no se han podido incluir, y que quedan planteadas, por tanto, como posibles mejoras futuras. Seguidamente, mencionamos algunas de las ideas que en algún momento han sobrevolado, aunque haya sido fugazmente, el desarrollo de este proyecto.

Integración en el Juez de ¡Acepta el reto!

La herramienta de extracción de pistas para envíos de problemas, se ha implementado como una aplicación autónoma, que dados el problema y la solución a probar, muestra la consiguiente pista, sin embargo, no se ha llegado a integrar en el Juez de Acepta el reto, debido a la falta de tiempo. Además, al integrarlo, se deberá adaptar la herramienta para tener mayor control sobre las ejecuciones, de manera que un código con intenciones maliciosas no pudiera ocasionar daños al Juez que pudieran provocar su parada. De igual forma, deberían tratarse también los veredictos de límite de

memoria superada y realizar las comprobaciones para averiguar si la salida generada hasta el momento del MLE era correcta o no, pues el entorno será proclive para ello.

Evaluación exhaustiva para TLE

Los veredictos que quizá sean más difíciles de interpretar en pos de conseguir una valoración de aceptado, posiblemente sean los que indican que se ha excedido el umbral de tiempo prefijado para resolver el problema. Es por eso que, hacer un análisis más exhaustivo, quizá pudiera otorgarnos información adicional para discernir si el TLE, es debido a un algoritmo no optimizado; a un bucle infinito; si nos quedamos muy cerca o muy lejos de la marca de tiempo establecida; si quizá el envío cumple perfectamente los requisitos del problema, pero debido a la carga del servidor en el momento en el que la recibe, éste tarda más de lo esperado, provocando así un Time Limit Exceeded; o si el veredicto es fruto del uso de alguna librería o método en concreto que resulte especialmente ineficiente, y demore por tanto la ejecución en exceso.

Integración con ejercicios Interactivos

Paralelamente al desarrollo de este trabajo, existen otros proyectos de distinta índole que permiten añadir determinadas características al juez de programación. Una de ellas consiste en la inclusión de ejercicios interactivos, en los que la resolución del problema se hace a base de una comunicación con el juez en la que se van obteniendo resultados intermedios hasta conseguir llegar a una solución final válida.

Dado el cambio de enfoque en este tipo de ejercicios, sería necesaria la inclusión de algunos veredictos adicionales, así como de pistas relacionados con los mismos. Además sería necesario acomodar el código pertinentemente para adaptarse a la forma de devolver los veredictos por parte del juez tras la ejecución de un ejercicio interactivo.

Pistas basadas en la comunidad

Cada problema plantea unas dificultades distintas, y hallar una solución válida puede suponer un reto considerable. Tanto es así, que quizá las pistas que el Juez devuelva pueden no ser suficientes para resolverlo, dependiendo de la combinación de tipo de problema y usuario que se plantee. Sin embargo, quizá el obstáculo que el usuario está enfrentando, ya haya sido solventado por otro usuario, que también necesitó algo de ayuda para conseguir su veredicto de aceptado, por lo que quizá, la mejor de las pistas sería recibir información de cómo ese usuario consiguió resolverlo.

En general, algo como "Tu amigo X, obtuvo un veredicto y una pista iguales que las tuyas justo antes de resolver el problema. Su solución, pasó de generar esta salida, a generar esta otra"; o "el N% de los usuarios que sufrieron los mismos errores que tú alcanzaron la solución modificando la salida que generaban en el caso Z".

Pistas en base a la dificultad del problema

Como ya se ha mencionado, en Acepta el Reto cohabitan problemas de distintas dificultades, por lo que parece buena idea adaptar las pistas al nivel de dificultad del problema, equilibrando el nivel de pistas y la dificultad, para que el hecho de recibir pistas, no suponga desmantelar por completo el planteamiento inicial del autor del problema. Esta característica, podría enfocarse más a un modo concurso o similar, en el que las pistas fueran distintas o directamente inexistentes.

Aparte de la integración en ¡Acepta el reto!, que obviamente es lo más apremiante, en mi opinión, y de cara a futuros desarrollos que continúen este trabajo, considero como de mayor relevancia de entre todas estas ideas, la integración con los ejercicios interactivos y la evaluación más exhaustiva de los casos con veredicto TLE, por constituir líneas de trabajo paralelas al presente proyecto, que podrían suponer una mejora importante para el juez. Asimismo, las pistas basadas en la comunidad, son un tipo de pista bastante interesante, pero que tomando como punto de partida el análisis presentado en las subsecciones anteriores, necesitaría de un arduo análisis tanto de las soluciones enviadas por los usuarios, como de las pistas obtenidas y de la secuencia de veredictos que obtienen los usuarios en su camino hasta el AC.

Conclusions

In the following lines, I will pose some of the lessons learned during the development of this project, both personally and in terms of collection and analysis of certain statistics. Finally, I will end with an exhibition of those objectives that are left raised in the future, by not being able to reach them this time.

Personal Rating

First, I would like to express the continuity quality of this project, which is an extension for my Final Project Degree in terms of functionality of a programming online judge. However, unlike that occasion, this time I changed sides in the project, to be more attached to the evaluator judge.

As for the project, it has managed to fulfill all the basic objectives that had been proposed at the beginning, respecting existing requirements.

Initially, I started from a set of applications and tools already implemented which I had to familiarize with. That, given the amount of existing code, was the first of the challenges faced, but fortunately I was able to overcome it and continue with the proposed work.

With this project, the teacher, and in the future the judge itself, is provided with an assessment tool able to give different types of useful clues to shipments sent with errors.

Error detection by a professor in code that is not his own, cannot always be trivial, especially depending on the implementation being subjected to evaluation. Especially for cases with dependence, it is possible that the way in which the solution is programmed hides this dependence and the fact of finding the cause of the problem costs an additional effort, however, automating the process, the time spent on it is reduced.

In addition, this functionality is one of the greatest shortcomings of the judge (and one of the most demanded by users), so the inclusion of this functionality can be assumed as a differentiating factor relative to other judges, as if you look back, we can remember that the appearance of hints in shipments is a weak point in many of the judges.

I would also like to comment on the close relationship between the codes for problems solutions proven and application behavior, because in my case have led me to the discovery of a large number of programming techniques, I had not seen before, for not being frequent to see them or for using an approach that is, at least, curious. There have been several occasions thanks to this kind of solutions, I have finished realizing that in certain very specific extreme

situations, my code did not cover all possible cases, which clearly has been another of the challenges that I have been in the position to face, and luckily overcome.

There is also another conclusion we can get, based on the collected statistics and analysis undertaken, which is that the clues are given, really help users as explained in the previous chapter.

It is true that in certain clues, the given information could be extended, but the reason for not doing so is no other than to maintain the philosophy of the judge on which we work, that is, maintain a certain level of difficulty so that it can be a challenge, but having at least one clue for being able to find what direction is the most appropriate to follow. This characteristic is, in my opinion, a double advantage, because apart from the help you offer the user, that affects equally in their success rate and ultimately also in their moral which will increase at the same rate as their use of the judge. This situation will gradually raise "the loyalty" from the user to the judge.

Therefore, I believe that this tool will be useful and will bring great benefits, especially in the teaching / educational field, as the use of the hints will be widespread among students, teachers and other participants of programming contests.

Future Work

Unfortunately, despite the time and effort spent on this project, there are certain features that are not possible to include, and are left, therefore, as possible future improvements. Then, we mention some of the ideas that have been considered at some point, even if it was fleetingly, during the development of this project.

Integration in ¡Acepta el reto!

The hint extraction tool for shipments of problems, has been implemented as a standalone application, that given the problem and the solution to be tested, shows the resulting clue, however, I have not gotten to integrate into the judge ¡Acepta el reto! due to lack of time. In addition, to integrate, you must adapt the tool to have greater control over the executions, so that a code with malicious intent cannot cause damage to the Judge that could cause his stop. Similarly, memory limit verdicts should also be covered and checks to see if the generated output until the MLE moment was correct or not should be performed too.

Comprehensive Assessment for TLE

The verdicts that may be more difficult to interpret for getting an assessment of accepted, are possibly those that indicate that the threshold time has been exceeded trying to solve the problem. That's why, doing a more thorough analysis could perhaps give us additional information to discern whether the TLE, is due to a non-optimized algorithm; is due to an infinite loop; if we were near or far from the timestamp established; if perhaps the shipping perfectly meets the requirements of the problem, but because of the server load at the time it received, it takes longer than expected, causing a Time Limit Exceeded; or if the verdict is the result of using any particular method or library particularly inefficient, and thus delay the execution excessively.

Integration with interactive exercises

Parallel to the development of this work, other projects for adding certain features to judge programming have been developed. One of them is the inclusion of interactive exercises, where the resolution of the problem is based on communication with the judge in which intermediate results are obtained until reaching a final valid solution.

Given the shift in the approach to this type of exercise, the inclusion of some additional verdicts would be necessary, as well as clues related to them. It would also be necessary to accommodate the code appropriately to suit the way of returning the verdicts by the judge after the execution of an interactive exercise.

Community-based hints

Every problem presents some different challenges, and find a valid solution can be a considerable challenge. So much that, perhaps the fact that the judge return hints may not be enough to solve it, depending on the combination of user and type of problem that arises. However, maybe the obstacle that the user is facing, has already been solved by another user, who also needed some help to get their verdict accepted, so perhaps the best hints would be to receive information on how that user managed to solve it.

In general, something like "Your friend X, obtained the same verdict and hints that you obtained just before solving the problem. His solution, passed from generate this output to generate this other "; or "N% of users who suffered the same mistakes that you have suffered, reached the solution by modifying the output generated in the Z case."

Hints based on the difficulty of the problem

As already mentioned, there are problems in iAcepta el reto! of various difficulties, and for this reason it seems a good idea to adapt the clues to the level of difficulty of the problem, balancing the level of hints and difficulty, for not destroying completely the initial approach of the author of the problem with the given hint. This feature may be more useful focusing in its application in a contest mode or similar in which the hints could be different or nonexistent directly.

Apart from the integration in iAcepta el reto!, which is obviously what is more urgent, in my opinion, and thinking about future developments to continue this work, I consider most important among all these ideas, the integration with interactive exercises and the most comprehensive assessment of cases with verdict TLE, for constituting parallel lines of work to this project, which could make a significant improvement to the judge. Community-based hints, are a type of very interesting hint, but taking the analysis presented in the previous chapter as a starting point, it would require a hard analysis of solutions sent by users, the obtained hints and the obtained sequence of verdicts that users get in their way to the AC.

Bibliografía

[1] G. L. McDowell, Cracking the Coding Interview: 150 Programming Questions and Solutions, CareerCup, 2011.

[2] «UVa Online Judge,» [En línea]. Available: https://uva.onlinejudge.org/. [Último acceso: 9 junio 2016].

[3] «CodeChef,» [En línea]. Available: https://www.codechef.com/. [Último acceso: 10 marzo 2016].

[4] «hackerearth,» [En línea]. Available: https://www.hackerearth.com/. [Último acceso: 10 marzo 2016].

[5] «Sphere online judge,» [En línea]. Available: http://www.spoj.com/. [Último acceso: 10 marzo 2016].

[6] «Codeforces,» [En línea]. Available: http://codeforces.com/. [Último acceso: 10 marzo 2016].

[7] «¡Acepta el reto!,» [En línea]. Available: https://www.aceptaelreto.com/. [Último acceso: 4 mayo 2016].

[8] «LeetCode Online Judge,» [En línea]. Available: https://leetcode.com/. [Último acceso: 10 marzo 2016].

[9] H. Kniber, Scrum and XP from the Trenches (Traducción de Ángel Medinilla), InfoQ, 2007.